The Traffic Assignment Toolboxes of Scilab

Pablo Lotito, Elina Mancinelli, Jean-Pierre Quadrat, Laura Wynter

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

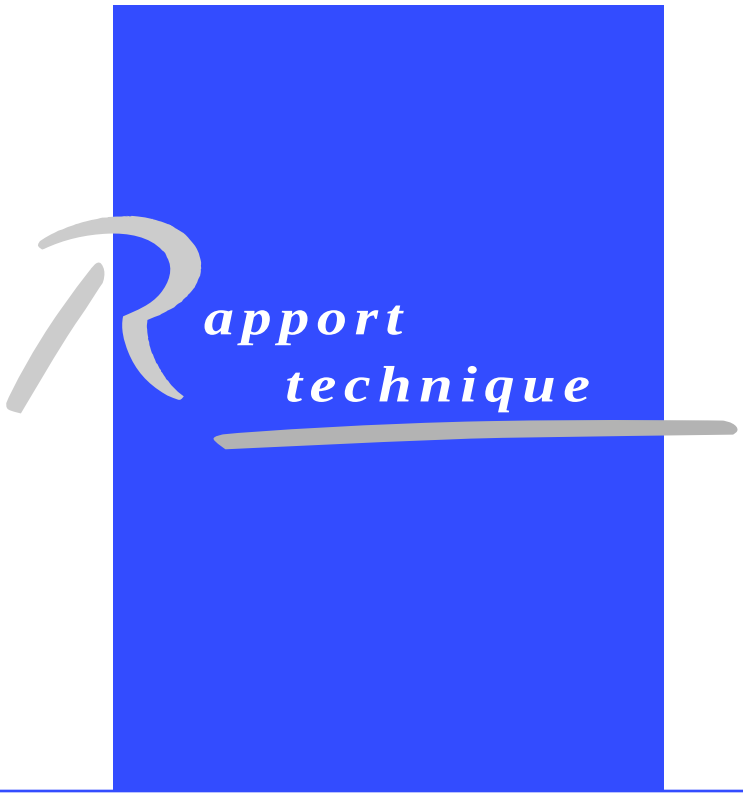
The Traffic Assignment Toolboxes of Scilab

Pablo Lotito — Elina Mancinelli — Jean-Pierre Quadrat — Laura Wynter

N° 0281

Mai 2003

THÈME 4



ISSN 0249-0803

The Traffic Assignment Toolboxes of Scilab

Pablo Lotito* , Elina Mancinelli†* , Jean-Pierre Quadrat* , Laura Wynter‡

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet Metalau

Rapport technique n° 0281 — Mai 2003 — 87 pages

Abstract: We present the two toolboxes of SCILAB dedicated to the traffic assignment. The first one, called CIUDADSIM, is devoted to the single class single mode case. The second one, called MCIUDADSIM, is concerned with the multiclass multimode case. After recalling the notions of Wardrop equilibrium and stochastic assignments the toolboxes are introduced and the manuals — describing completely all the functions — are given.

Key-words: traffic assignment, Wardrop equilibrium, logit

* INRIA-Rocquencourt
† CONICET
‡ IBM

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 —Télécopie : +33 1 39 63 53 30

Les Boîtes à Outils Trafic de Scilab

Résumé : Nous présentons les deux boîtes à outils de SCILAB dédiées à l'affectation de trafic. La première, appelée CIUDADSIM, est consacrée à l'affection dans le cas d'une seule classe d'usagers et d'un seul mode de transport. La deuxième, appelée MCIUDADSIM, est dédiée à l'affectation dans le cas de plusieurs classes d'usagers chacun pouvant choisir entre plusieurs modes de transport. Après avoir rappelé les notions d'équilibre de Wardrop et les méthodes d'affectation stochastique les boîtes à outils sont présentées puis les manuels de références — décrivant toutes les fonctions des boîtes à outils — sont donnés.

Mots-clés : affectation de trafic, équilibre de Wardrop, logit

1 Introduction

This report describes two toolboxes of Scilab dedicated to the traffic assignment : – the first one CiudadSim is concerned with the single class single mode case, – the second one MCiudadSim with the multimode multiclass case.

The purpose of traffic assignment is to compute the flows on the routes of a transportation network taking into account the arc congestions. In the deterministic case the travel times are deterministic and known by the users in the stochastic case the users make an error in the perception of the travel time, but the distribution of this error is known. The network is used by several class of users. Each class has its own criterion. Several transportation modes are available for each class of users. In stationary regime, equilibriums are defined, they determine the arc flows, the toolboxes compute these equilibriums.

At address http://www.certu.fr/transport/s_pages/modelisation/logiciel.htm one can find an exhaustive list of specialized softwares devoted to the solution of this problem. But none is free and moreover very often the assignment methods used are not described precisely. Here we discuss a free toolbox of Scilab — which is a free scientific numerical computation software very similar to Matlab —. The main assignment algorithms (All or Nothing, Frank Wolfe, Desaggregated Simplicial Decomposition, Logit, Probit) are available in this toolbox. They are coded partly in Scilab partly in C and the sources are available. The network can be visualized and edited using Scilab facilities. An interface with the Mapinfo geographic data base is also provided.

In a first part the equilibriums, main algorithms and examples are described in the two cases (single class and multiclass) then the complete reference manuals are given. These manuals are also available on line in the help of Scilab as soon as the toolboxes are loaded.

2 Single Class Traffic Assignment

Given a transportation network $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ and a set $\mathcal{D}$ of transportation demands from $o \in \mathcal{N}$ to $d \in \mathcal{N}$ the traffic assignment problem consists in determining the flows f_a on the arcs $a \in \mathcal{A}$ of the network when the times t_a spent on the arcs a are given functions of the flows f_a. These flows are defined as a solution of an equilibrium. We can distinguish the deterministic equilibriums from the stochastic ones.

2.1 Wardrop Equilibrium

In the deterministic case two important equilibriums have been defined : – the system equilibrium where the flow is determined by the minimization of the total time spent by all the users of the network, – the user equilibrium — called sometimes Wardrop equilibrium — where each user minimizes his time spent in the network. At the Wardrop equilibrium we have : *"for all pair of nodes o and d in the network, the times spent on all the actually used routes from o to d are the same and are smaller than the times spent on the unused routes"*.

Denoting by $\mathcal{C}$ the set of origin-destination pairs, R_{od} the set of routes from o to d, d_{od} the demand from o to d, f_r the flow on the route $r \in R_{od}$, F_a the arc flow on $a \in \mathcal{A}$, t_r the travel time on the route $r \in R_{od}$, t^*_{od} the smallest travel time of the routes in R_{od}, the *Wardrop Equilibrium* is defined precisely by the following equations

$$\min\{f_r, t_r - t^*_{od}\} = 0$$

$$\sum_{r \in R_{od}} f_r = d_{od}\ , \quad \forall r \in R_{od},\ \forall od \in \mathcal{C}\ .$$

This system of equations is the optimality condition of two optimization problems, the nodes-arcs and the arcs-routes variational formulation. The *nodes-arcs variational formulation* of the Wardrop Equilibrium is :

$$\min_f \sum_a \int_0^{F_a} t_a(q)\, dq, \quad A f^{\bullet}_{od} = d_{od}, \quad f \geq 0\ .$$

where A is the $m \times n$ incidence matrix nodes-arcs associated to the network, f^a_{od} is the flow on the arc a of the commodity od, $F_a = \sum_{od} f^a_{od}$ is the total flow on the arc a and $t_a(q)$ denotes the travel time if the total flow on this arc is q.

The *arcs-routes variational formulation* of the Wardrop Equilibrium

$$\min_f \sum_a \int_0^{F_a} t_a(q) dq\ ,\ F_a = \sum_{r \in R^a} f_r$$

$$\sum_{r \in R_{od}} f_r = d_{od}\ ,\ f \geq 0\ ,$$

where R^a denotes the set of all the routes of the network using the arc a.

The main drawback of this Wardrop equilibrium is that each traveler is supposed to have a perfect information about the total network. In more realistic formulations the user minimizes the perceived travel time. This perceived travel time is the effective travel time with an additive error. If the probabilistic distribution of the errors is supposed to be known we can define stochastic equilibriums.

2.2 Logit Assignment

At least two kinds of stochastic equilibrium have been studied intensively in the literature according to the distribution of the error in the perceived travel time. In the *probit assignment* problem the error is supposed to be Gaussian. But, because $\mathcal{N}\{X > x\}$ for $\mathcal{N}$ a Gaussian law, is not known explicitly the computation of probit assignment is difficult and

it is done by Monte Carlo methods. When the distribution of the error is a Gumbel distribution $\mathcal{G}\{X < x\} = e^{-e^{-\mu x}}$ the probability of choosing a route $r \in R_{od}$ can be computed explicitly. It is given by

$$\forall r \in R_{od} \quad p_r = \frac{e^{-\mu t_r}}{\sum_{r \in R_{od}} e^{-\mu t_r}} , \tag{1}$$

where t_r denotes the time spent on the route r. This assignment is called *logit assignment*. The only interest of the Gumbel distribution comes from the facility to compute the probability of the maximum of two independent random variables and from its shape close to the normal law distribution.

The logit assignment defines the only distribution satisfying the efficiency principle [6] that is : *"an event of flows on the available road having a smaller average time than another one has a greater probability to appear"*. In [6] other equivalent properties of the logit distribution (which is in fact the Gibbs distribution of mechanical statistics) are given. In particular it minimizes the entropy among all the flow distributions having the same average time. The free parameter μ is a degree of stochasticity. Its inverse is the temperature in statistical mechanics.

The first difficulty appearing with computing logit assignment is that, as soon as it exists a circuit in the graph, it will exists a pair od such that the number of routes in R_{od} is infinite and therefore it may be difficult to compute the logit distribution(1). Dial [5] proposes to consider only efficient paths r that are paths $r = op_1p_2 \cdots d$ such that, given an origine o and a destination d, the travel times from a node p_i of the path to the destination are always decreasing with the nodes p_i met, and the travel times from the origine to a node of the path are increasing with these nodes. Then denoting $\mathcal{G}^{od} = (\mathcal{N}, \mathcal{A}^{od})$ the new graph obtained by eliminating the arcs for which this increasing and decreasing property is not satisfied and $\mathcal{O}^{od}$ [resp. $\mathcal{D}^{od}$] the corresponding incidence node-arc-origin [resp. node-arc-destination] matrix we can define the *transition weight matrix*

$$W^{od} = \mathcal{O}^{od} T (\mathcal{D}^{od})' ,$$

where T is diagonal matrix $T_{aa} = e^{-\mu t_a}$, $a \in \mathcal{A}^{od}$. We have

$$W^{od}_{ij} = T_{aa}, \quad \forall a \in \mathcal{A}^{od} .$$

There exists a node numbering such that the matrices W^{od} are strictly upper diagonal and therefore they are nilpotent. Thus $(W^{od})^*$ exists, where $N^* = \sum_{i=0}^{\infty} N^i$. Then the logit assignment on efficient path is given by :

$$F_{ij} = \sum_{od \in \mathcal{D}} d_{od} (W^{od})^*_{oi} W^{od}_{ij} (W^{od})^*_{jd} / (W^{od})^*_{od} . \tag{2}$$

Indeed

- $(W^{od})^*_{od}$ is the total weight of all the efficient path from o to d,

- $(W^{od})^*_{oi} W^{od}_{ij} (W^{od})^*_{jd}$ is the total weight of the efficient paths that use the arc ij,
- $(W^{od})^*_{oi} W^{od}_{ij} (W^{od})^*_{jd} / (W^{od})^*_{od}$ is the probability for a path from o to d of using the arc ij for the logit distribution on the efficient paths.

In fact, Dial in [5] does not give the formula (2) but an algorithm in two phases to compute the contribution of one demand to the flow on an arc. Mainly in the first phase he computes $(W^{od})^*_{oi} W^{od}_{ij}$ and in the second phase he computes $(W^{od})^*_{jd} / (W^{od})^*_{od}$ from which he deduces the flow. Moreover the exact weights used by Dial are not the ones given here but the weights $e^{-\mu \tau_{ij}}$ with τ_{ij} defined by (5).

A first improvement, given in [13], consists in considering as efficient paths the paths which have an increasing time from the origin and not restricting them to have also a decreasing time to the destination. With this new definition of efficient paths we build new graphs $\mathcal{G}^o = (\mathcal{N}, \mathcal{A}^o)$, incidence matrices $\mathcal{O}^o$, $\mathcal{D}^o$ transition weight matrix W^o. The corresponding flow has a formula analogous to (2) but now can be written

$$F_{ij} = \sum_{o \in \mathcal{N}} (W^o)^*_{oi} W^o_{ij} (W^o)^*_{j.} D^o \tag{3}$$

with

$$D^o_d = \begin{cases} d_{od} / (W^o)^*_{od} & \text{if } od \in \mathcal{D} \\ 0 & \text{else} \end{cases} .$$

This formula shows that by this way the complexity can be reduced of one order in the size of the problem.

Another improvement proposed by Bell [1] is to consider all the paths, not only efficient paths, by remarking that we have not to enumerate all the paths but to be able to compute star of matrices. But, $W^* = (I - W)^{-1}$ is well defined as soon as the spectral radius of W, denoted $\rho(W)$, is smaller than 1. Moreover with this point of view it is not necessary to compute the minimal time from the origin to each node to define the transition weight matrix associated to efficient paths. Therefore, denoting by W the transition weight matrix for the original graph $\mathcal{G}$, we have :

$$F_{ij} = \sum_{o \in \mathcal{N}} (I - W)^{-1}_{oi} W_{ij} (I - W)^{-1}_{j.} D^o \tag{4}$$

with

$$D^o_d = \begin{cases} d_{od} / (I - W)^{-1}_{od} & \text{if } od \in \mathcal{D} \\ 0 & \text{else} \end{cases} .$$

With this point of view we must choose μ large enough such that $\rho(W) < 1$ (this is always possible because the entries of W are positive and go to zero when μ goes to $+\infty$).

We can avoid this last difficulty by building a Markov chain on $\mathcal{G}$ depending on a parameter μ whose trajectories converge to the minimal time trajectories when μ goes to $+\infty$.

Let us consider the transition matrices

$$M_{ij}^d = e^{-\mu\tau_{ij}} / \sum_j e^{-\mu\tau_{ij}} \ ,$$

with

$$\tau_{ij} = t_{ij} + t_{id}^* - t_{jd}^* \ , \tag{5}$$

where t_{id}^* denotes the minimal time to go from i to d. This gives a logit type probability of deviating from the optimal trajectory. Therefore here we consider a multidecision process where at each step we have to choose between trajectories composed of an arbitrary admissible link followed by an optimal route to node d. Based on this type of decision the flows on arcs can be computed explicitly. We have

$$F_{ij} = \sum_d D_d (I - M^{\natural d})^{-1} M^{\flat d} M_{ij}^d \ , \tag{6}$$

where D_d is the row-vector of the demand from any node to d, $M^{\natural d}$ denotes the matrix obtained by skipping column and row d in matrix M^d and $M^{\flat d}$ the column-vector obtained by extracting column d of M^d and skipping entry d.

2.3 Stochastic Equilibrium

In the previous section about stochastic assignment we have supposed implicitly that the travel time does not depend on the flow. If it is not the case we have to solve an implicit equation. For example in the third case, given the function $t_a(F_a)$ for $a \in \mathcal{A}$ we have to solve the equation

$$F = \mathcal{F}(F) \tag{7}$$

with

$$[\mathcal{F}(F)]_{ij} = \sum_{p \in \mathcal{N}} (I - W(F))^{-1}_{oi} W(F)_{ij} (I - W(F))^{-1}_{j.} D^o(F) \ ,$$

where $W(F)$ is the weight transition matrix which depends on the flows.

This equilibrium admits an arc-route variational formulation :

$$\min_f \sum_r f_r \log(f_r) + \sum_a \int_0^{F_a} t_a(q) \ , \tag{8}$$

under the constraints :

$$F_a = \sum_{r \in R^a} f_r, \ \sum_{r \in R_{od}} f_r = d_{od}, \ f \geq 0 \ .$$

The fixed point algorithm

$$F^{n+1} = a_n F^n + (1 - a_n)\mathcal{F}(F^n) \ ,$$

with typically $a_n = (n-1)/n$, is used to solve (7). The convergence of this method can be derived from the variational formulation. Indeed $\mathcal{F}(F^n)$ is the solution of

$$\min_f \sum_r f_r \log(f_r) + \sum_a t_a(F_a^n)(F_a - F_a^n) \ ,$$

which is a partially linearized version of problem (8). Then using the convexity of this last problem $\mathcal{F}(F^n) - F^n$ is a descent direction of problem (8). The divergent series method is used because it is difficult to compute the value of the criterion, indeed it needs the evaluation of the flows on all the routes (which may be in infinite number).

3 Introduction to the Single Class SCILAB Toolbox (CIUDADSIM)

The Traffic Network toolbox of SCILAB called CIUDADSIM is intended to study the assignment problem in traffic networks that is to compute the traffic on the links of a network knowing the transportation demands for some couple of origin and destination in the given network. To do that CIUDADSIM adds dedicated functions and data structures to SCILAB. It uses Scigraph (a toolbox of SCILAB) to visualize and edit the network.

CIUDADSIM is a SCILAB contribution which is free and available at [4]. It needs : – SCILAB a free Matlab like general purpose scientific software available at [11], – SCIGRAPH a SCILAB toolbox for visualizing graph available at [12], – MAXPLUS a SCILAB toolbox for max plus arithmetic available at [10].

CIUDADSIM contains a database called `%net` which contains all the given data – geographic definition of the network, – travel demands (*commodities*), – travel time as functions of congestion (link performance functions). It contains also all the computed data – the aggregated and disaggregated (with respect to the commodities) flows on the links, – the times spent on each link at equilibrium.

The function `TrafficExample` builds the database for some examples.

The function `ShowNet` visualizes the net in an editable window — it is possible to edit the database using the menus of this visualizing window —.

A SCILAB function `TraffigAssig` computes traffic assignments (Wardrop equilibrium, logit and probit equilibrium) using various algorithms. The results are written in some fields of `%net`. The assignment obtained can be visualized using `ShowNet`. The widths of the links indicate their congestion degrees.

4 Single Class Database: %net

A traffic network has three well distinguished parts. The nodes, the links and the demands. The data type for each of those parts will be as well different. For example for the nodes we will only need their coordinates but for the links we will need together with their head and tail nodes, the corresponding link performance functions. We will consider the O-D pairs together with the corresponding traveling demands as arcs directed from the origin to the destination with the demand as associated weight.

All this data are collected in a Scilab typed list called `%net` which play the role of a geographic database.

The link performance function of each link is stored in the database as a function of the flow. It can be chosen among :

1. $t(f) = t_0 + \frac{m}{c} f + m \max(0, f - c)^{\beta}$
2. $t(f) = t_0 \exp\left(\frac{f}{c} - 1\right)$
3. $t(f) = t_0 \, 2^{\left(\frac{f}{c} - 1\right)}$
4. $t(f) = t_0 \left(1 + 0.15 \times \left(\frac{f}{c}\right)^m\right)$
5. $t(f) = t_0 + \log\left(\frac{c}{c-f}\right)$
6. $t(f) = \beta - c\frac{t_0 - \beta}{f - c}$
7. $t(f) = t_0 + c\, f + m\, f^{\beta}$

These functions depend on the four parameters t_0, c, m, β, where – t_0 is the free-flow travel time, which is the travel time at zero flow, – c is the practical capacity of the link, which is a measure of the quantity from which the travel time will increase very rapidly.

The method to compute the assignment is given in the field `algorithm` of `%net.gp`, the default one is the DSD described below. The main results of the assignment are stored in the fields `%net.links.flow` and `%net.links.time`.

The function `TrafficExample` gives a way to build some predefined database. For example to build the regional network of Chicago we use the instruction

```
%net=TrafficExample('Chicago').
```

Then `%net` is visualized by the instruction `ShowNet()` which shows the graph (Figure 1) in an editable graphic window of Scilab.

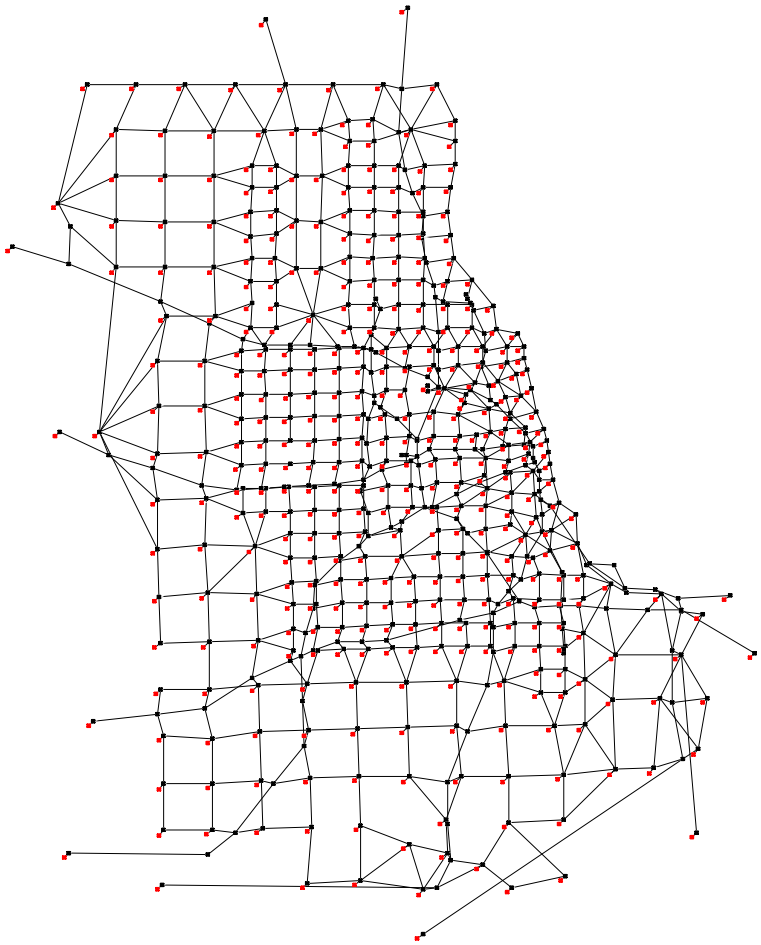


Figure 1: Chicago Net

5 Single Class Examples

The assignment is done by the instruction `TrafficAssig()`. The default algorithm is the `'DSD'` which is a newton method to compute a Wardrop equilibrium. We can change the algorithm by changing `%net.gp.algorithm`. If we choose a logit algorithm we have to specify the level of stochasticity by changing the default value of `%net.gp.theta`. A discussion and comparison of the respective interests of the algorithms are given in the section "Numerical Experiments".

In the toolbox there are some available examples. In the following is given only a sampling of the possibility. Typically to make an assignment for an existing example we have to write the following instructions in the SCILAB window :

```
%net=TrafficExample('Example')
TrafficAssig()
ShowNet()
```

where `'Example'` is the name of an available example. If a new example has to be entered the best way is to build `%net` with `MakeNet`. It is also possible to edit a new net with the Scigraph window.

5.1 Regular City

The instruction

```
%net=TrafficExample('Regular City',hs,vs,nd)
```

generates a regular city whose graph is a `hs` x `vs` grid. The link travel time functions data is the same for all links and is defined by the parameters `t0=ca=ma=ba=1` and the link performance model 3 therefore link performance function is $c = t_0(1 + 0.15f)$.

It is possible to generate 4 different types of Regular City depending on parameter `nd`.

1. `nd=[ ]`. The net has all possible O-D pairs all of them with demand value equal to 1. The first picture in Figure (2) is shown using the function `ShowLinks` because there

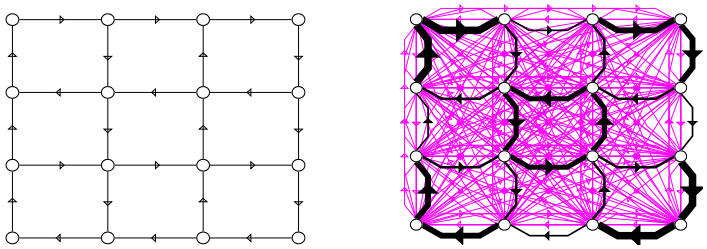


Figure 2: Non-assigned and Assigned Regular City Net `TrafficExample('Regular City',4,4)`

   are to many demands.

2. `nd=1`. There is only one O-D pair joining the most distant nodes with demand equal to 1.

3. `0<nd<1`. The parameter *nd* represents the density of O-D pairs present in the net and the demand is equal to 1.

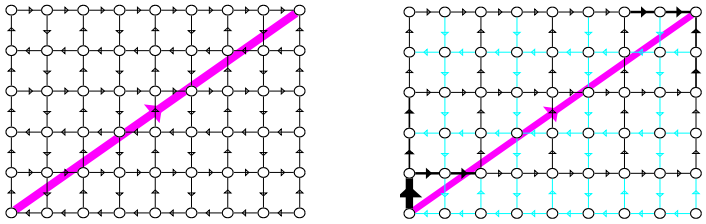


Figure 3: Non-assigned and Assigned Regular City Net `TrafficExample('Regular City',6,9,1)`

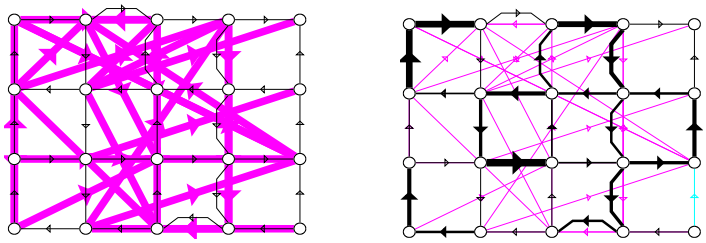


Figure 4: Non-assigned and Assigned Regular City Net `TrafficExample('Regular City',4,5,0.09)`

4. `1<nd`. There are `nd` random demands generated with value uniformly distributed between 0 and 15.

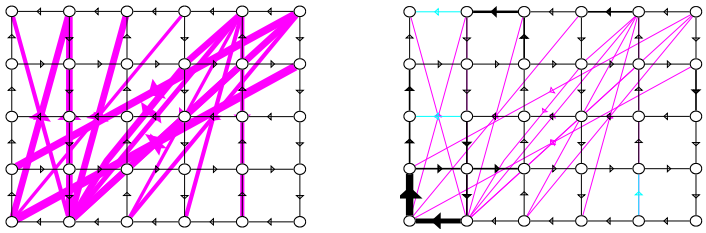


Figure 5: Non-assigned and Assigned Regular City Net `TrafficExample('Regular City',5,6,15)`

5.2 Sioux Falls Net

It is an example with 24 nodes 72 links and 528 demands with link performance function of the form $c = t_0 + m_a f^4$ with t_0 between 0.02 and 0.1 and m_a between 10^{-8} and 2×10^{-5}. See Figure (6).

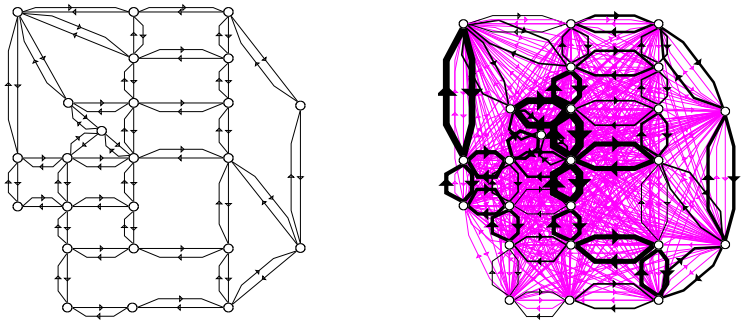


Figure 6: Non-assigned and Assigned Sioux Falls Net `TrafficExample('Sioux Falls')`

5.3 Steenbrink Net

The Steenbrink example (see [14]) is a network with 9 nodes 36 links and 12 demands with link performance function of the form $c = t_0 + m_a f$ with t_0 between 0 and 1 and m_a between 3×10^{-4} and 2×10^{-3}.

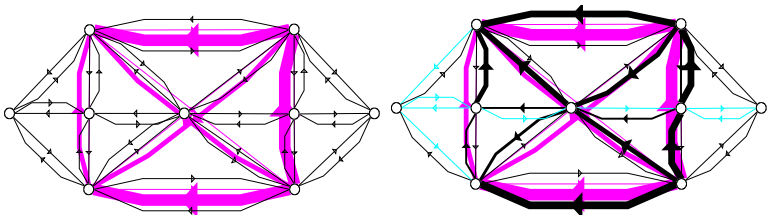


Figure 7: Non-assigned and Assigned Steenbrink Net `TrafficExample('Steenbrink')`

6 Single Class Assignment Algorithms

Let us describe briefly the algorithms used to compute Wardrop equilibrium which have been implemented. Logit assignment correspond to explicit formula given in the section "Traffic

Assignment", the corresponding equilibrium uses a method of successive average described in the subsection "Stochastic Equilibrium". The corresponding algorithms are implemented in the functions

- `MSA` method of successive average in deterministic case,
- `LogitNE` method of successive average in logit case,
- `MSASUE` method of successive average in probit case.

6.1 All or Nothing algorithm : `AON`

When the times spent on the arcs do not depend on the flows on the arcs, the problem may be reduced to compute the routes with the smallest travel time for each pair od in $\mathcal{C}$.

6.2 Incremental Loading Heuristic : `IA`

The demand is split in a sum of elementary demands. Each elementary demand is assigned using the AON algorithm after a fresh evaluation of the travel time based on the previous loadings.

The assignment obtained by this method does not correspond in general to a Wardrop Equilibrium.

6.3 Newton Method for the arcs-nodes variational formulation : `WardropN, ('NwtArc')`

Using the nodes-arcs formulation, introducing the dual variable V associated to the conservation law written $AF = D$, and denoting $1/2F'RF$ the quadratic criterion to be minimized, we obtain the optimality conditions :

$$AR^{-1}\max(A'V,0) = D\,,\quad F = R^{-1}\max(A'V,0)\,.$$

As soon as the network is strongly connected the first equation defines V up to a constant. It can be computed efficiently by a Newton method which is not globally convergent. F is unique and may be deduced from V using the second equation.

Using relaxation algorithm this method had been adapted for general costs to the multicommodity case.

It seems to be efficient when the number of commodities is not very large and when the flow depending part of the travel time dominates its fixed part.

6.4 Frank-Wolfe algorithm : `FW`

The nodes-arcs formulation is used.

The current assignment is improved according to :

- Given the flows F_a, the travel times $t_a(F_a)$ are computed and a AON assignment F' is done.
- The new assignment becomes $\lambda F + (1-\lambda)F'$:

$$\min_{1\geq\lambda\geq 0}\left(\sum_a c_a\left(\lambda F_a + (1-\lambda)F'_a\right)\right),\quad c_a(F_a) = \int_0^{F_a} t_a(q)\,dq\ .$$

It is the most common used algorithm. It may be slow because it is a first order method but it is not very memory consuming therefore it can be applied to large networks.

6.5 Disaggregated Simplicial Decomposition : DSD

This algorithm have been proposed by Larsson-Patriksson [8].

The nodes-routes formulation is used.

- At starting time, all the demand of a commodity is assigned to only one route by an AON algorithm.
- At each iteration, for all commodity, the shortest route (based on the current flow) is added to a set of memorized routes.
- The improved assignment is obtained by optimizing the flow on the memorized routes by a diagonalized Newton method.

Using the variables λ_{odr} which gives the proportion of the demand assigned to the route r for the commodity od we have

$$f_r = \lambda_{odr}\, d_{od}\,.$$

the arc flow on a is :

$$F_a = \sum_{od} d_{od} \sum_{r\in R^a_{od}} \lambda_{odr}\ .$$

The cost function becomes :

$$T\left(\lambda\right) = \sum_{a\in A} c_a\left(\sum_{od} d_{od} \sum_{r\in R^a_{od}} \lambda_{odr}\right)\ .$$

To compute the descent direction μ a second order approximation of the cost T is made:

$$T\left(\mu\right) \approx T\left(\lambda\right) + DT\left(\lambda\right)\left(\mu-\lambda\right) + \left(\mu-\lambda\right)'\, D^2T\left(\lambda\right)\,\left(\mu-\lambda\right)\ ,$$

where

$$[DT(\lambda)]_{odr} = d_{od} \sum_{a \in r} c'_a(F_a), \quad [D^2T(\lambda)]_{odr,o'd'r'} = d_{od}\, d_{o'd'} \sum_{a \in r \cap r'} c''_a(F_a) .$$

Then the new assignment is obtained by optimizing the quadratic diagonal approximation of the cost, that is taking only the diagonal part of the Hessian :

$$[D^2T(\lambda)]_{odr,odr} = d_{od}^2 \sum_{a \in r} c''_a(F_a) .$$

and solving the linear search :

$$\min_{\rho \geq 0} T(\lambda + \rho\mu) ,$$

where λ denotes the current assignment.

7 Single Class Numerical Experiments

All the implementations of the algorithms for this study where on a Pentium III 448.623 MHz computer with 384 MB of internal memory and a 7.4 GB hard disk. The algorithm were tested on networks known from the literature and over regular network of different sizes. In the tables time is expressed in seconds.

7.1 Deterministic Case

All the numerical examples shown here have some common properties. The best values for the objective function are reached with the DSD algorithm and at the same time it is the most memory demanding algorithm because it is a route-based algorithm. At each step route informations are stored using the back node arrays generated in the shortest path procedure.

The first example is the Steenbrink network. We observe here that DSD is the most efficient algorithm, it reaches to a better value of the cost function in a time which is largely smaller than the time needed by the other three algorithms.

Algorithm	Iter	Time	Cost	Mem. used	Rel. Gap
DSD	5	0.29	1.6957675e+04	2.7e+03	8.245e-05
FW	4325	38.75	1.6958960e+04	2.6e+04	9.995e-05
MSA	4494	19.89	1.6959068e+04	2.2e+04	9.853e-05
NwtArc	30	13.80	1.6957675e+04	2.1e+03	6.475e-05

Table 1: Steenbrink, nodes=9, links=36, demands=12

The second example is the well known numerical example of Sioux Falls. In this case it is not possible to apply the NwtArc algorithm. Here again DSD is the best algorithm taking into account computational time and accuracy, but it is also the most memory demanding.

Algorithm	Iter	Time	Cost	Mem. used	Rel. Gap
DSD	10	1.58	4.2313361e+01	6.3e+04	7.656e-05
FW	1789	23.48	4.2316005e+01	1.4e+04	9.999e-05
MSA	13933	181.24	4.2317409e+01	7.1e+04	9.999e-05

Table 2: Sioux Falls, nodes=24, links=72, demands=528

The following problem is a variant of Chicago Sketch.

Algorithm	Iter	Time	Cost	Mem. used	Rel. Gap
DSD	3	127.40	1.6753004e+07	4.7e+06	5.499e-04
FW	33	153.40	1.6753136e+07	4.9e+05	5.781e-04
MSA	137	408.09	1.6757105e+07	1.9e+05	5.489e-04

Table 3: Chicago, nodes=546, links=2176, demands=93135

We can see in the following example the performance of the 4 available algorithms to make the assignment in the case of few OD pairs. In the first table the difference between the values of the cost function for each algorithm are smaller than $1.7e-5$ but the difference in the computational times is significant.

Algorithm	Iter	Time	Cost	Mem. used	Rel. Gap
DSD	28	3.14	2.854915580e+01	1.4e+04	1.353e-04
FW	220	12.55	2.854917233e+01	3.7e+03	5.011e-05
MSA	258	2.86	2.854916956e+01	3.4e+03	9.140e-05
NwtArc	3	0.63	2.854917270e+01	9.3e+03	2.395e-08

Table 4: Regular City, nodes: 225, links:420, demands:1

The algorithm 'NwtArc', is the most efficient one in the case of a unique OD pair but it becomes less efficient when the demand number increases as we can see in table 5. The computation are made for a 10×10 regular network.

To illustrate the performance of the DSD algorithm compared to the FW and the MSA algorithms, we present results of the algorithms applied to a regular 8×8 network, whose sizes are: 64 nodes, 112 links, and 4032 OD pairs. See table 6.

Demand Number	Algo	Step	Time	Cost	Memory Used	Relative Gap
1	DSD	11	0.59	1.8470456e+01	5.1e+03	6.567e-05
	NwtArc	2	0.24	1.8470467e+01	4.6e+03	1.917e-06
2	DSD	14	0.92	3.7870531e+01	5.6e+03	5.010e-05
	NwtArc	6	0.71	3.7870529e+01	4.8e+03	8.141e-05
10	DSD	12	1.27	1.6139485e+02	1.4e+04	4.777e-05
	NwtArc	8	4.49	1.6139488e+02	5.7e+03	8.415e-05
100	DSD	13	5.56	2.1215788e+03	7.2e+04	5.197e-05
	NwtArc	15	97.33	2.1215787e+03	1.6e+04	3.331e-04
900	DSD	15	27.12	6.403253421e+04	2.9e+05	5.112e-04
	NwtArc	15	1105.02	6.403153657e+04	1.0e+05	2.870e-04
1000	DSD	15	31.61	7.607687457e+04	3.2e+05	8.147e-04
	NwtArc	15				

Table 5: Performance of the NwtArc algorithm

Algorithm	Iter	Time	Cost	Mem. used	Rel. Gap
DSD	6	135.67	5.0910377e+05	3.02e+05	1.2e-05
FW	3000	158.14	5.0911997e+05	2.7e+04	5.3e-05
MSA	3000	136.28	5.0912299e+05	2.4e+04	5.1e-05

Table 6: Regular, nodes=64, links=112, demands=4032

In Figure (8), the logarithm of the relative errors obtained for each algorithm are plotted again the CPU time also in logarithmic scale. The relative errors are computed with reference to the best value found for the cost function with the DSD algorithm with a precision of 10^{-9}. The appearance of Figure (8) is the expected one, since FW and MSA are both first order algorithms so they have a similar behavior decreasing like $1/t$ while the second order algorithm, DSD, decrease as a power of $1/t$.

7.2 Stochastic Case

For the case of stochastic assignment, we present here some results concerning the relation between the different type of logit algorithms. We consider the example of a 6×6 regular network with all possible OD pairs. In table (7) we show the results for the computation of the stochastic equilibrium. If parameter μ is very large, the perception error is small and users will tend to select the minimum measure travel-time path, if μ is small the share of flow on all paths will be equal. We can see this in Figure (9) where the computation were made with a logit-dial algorithm . The left graphic represents the flow assignment obtained

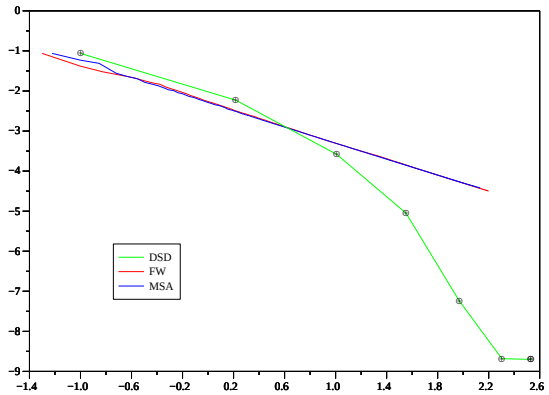


Figure 8: Relative error versus computation time (in logarithmic scale) for the FW (red), MSA (blue) and DSD (green) algorithms.

Algorithm	Iter	Time	Cost	Mem used	Error
Logit DE	147	41.27	6.2391744e+04	1.2e+05	9.922e-04
Logit BE	147	1.67	6.2391775e+04	1.2e+05	9.923e-04
Logit MDE	132	36.41	6.2390267e+04	1.2e+05	9.921e-04
Logit MBE	132	34.52	6.2390281e+04	1.2e+05	9.921e-04

Table 7: Stochastic Equilibrium: logit type algorithm comparison

with a $\mu = 1000$, this flow distribution is the same that is found using the AON algorithm. The graphic in the right is obtained with $\mu = 1/1000$, all efficient path are used and the flow is uniformly distributed among those paths.

In next example we consider a regular city with a unique demand, in this case all the possible paths joining the origin and destination nodes have the same cost. If we perform an assignment using the AON algorithm, all the demand flow is assigned to one path. In the other hand if a logit-type algorithm is used all the paths of equal cost are used.

Finally we present the results of computing the stochastic equilibrium for a large value of parameter μ with the method of successive average and a Bell-type logit, the obtained flow distribution is very similar to the distribution obtained by a deterministic assignment performed with the DSD algorithm. The maximum difference flow value is of the same order than the required precision (see Figure 11).

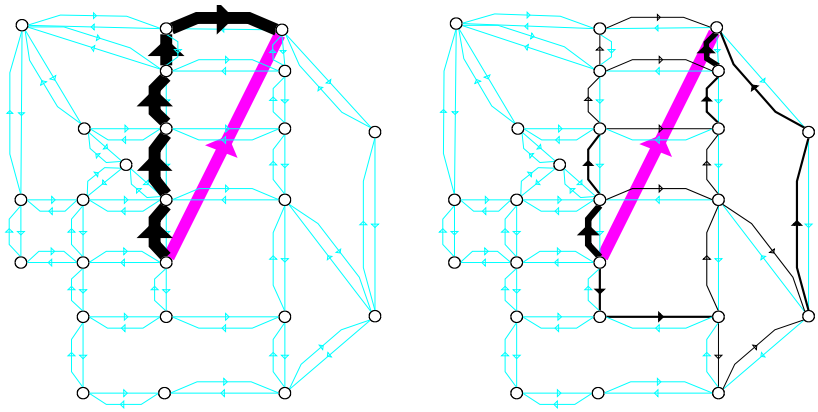


Figure 9: Logit assignment with $\mu = 1000$ and $\mu = 1/1000$

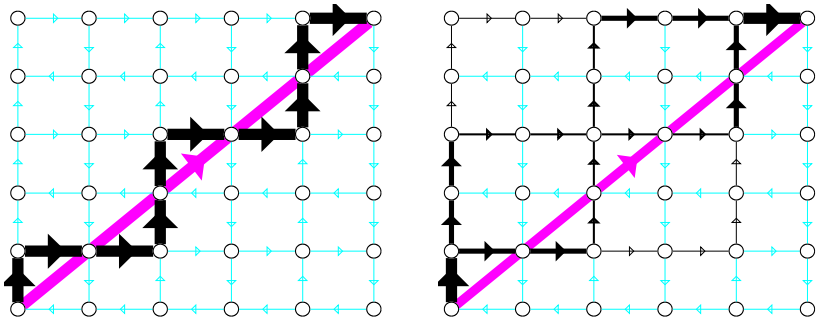


Figure 10: AON and Logit assignment in a regular network

Algorithm	Iter	Time	Cost	Mem used	Error
Logit BE	147	1.67	6.2391775e+04	1.2e+05	9.923e-04
DSD	3	2.08	6.2379125e+04	5.6e+04	3.666e-05

Table 8: Stochastic and deterministic equilibriums

8 Multiclass Traffic Assignment

In the multiclass multimode case we consider that the demands of traffic are grouped into classes of users with the same perception of costs (price and value of time). Then, the users of each class, can choose among different modes of transport. We assume that the

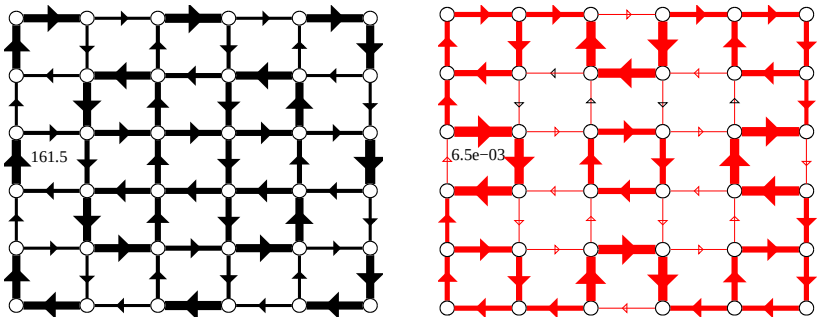


Figure 11: Stochastic equilibrium assignment and difference with a deterministic equilibrium assignment

users choose only one mode for the entire trip. As the modal choice is an important result variable we define $d_{od}^{k,m}$ the part of the demand d_{od}^{k} that uses the mode m.

For each class k, mode m and OD pair od we have in general many possible routes that we put in the set $\mathcal{R}_{od}^{k,m}$ and for each route r in $\mathcal{R}_{od}^{k,m}$ we call $f_r^{k,m}$ the associated flow. The first condition that the flows must verify is the demand satisfaction:

$$\sum_{m\in\mathcal{M}_k}\sum_{r\in\mathcal{R}_{od}^{k,m}} f_r^{k,m} = \sum_{m\in\mathcal{M}_k} d_{od}^{k,m}\,, \tag{9}$$

where $\mathcal{M}_k$ is the set of modes that class k can choose among.

As the congestion is modeled in terms of the arc flows we define the flow over the arc a as $f_a^{k,m}$ and it can be computed as

$$f_a^{k,m} = \sum\{f_r^{k,m} | a \in r \in \mathcal{R}_{od}^{k,m}, od \in \mathcal{D}^k\}\,. \tag{10}$$

8.1 Non congested assignment

8.1.1 Deterministic case

We consider that the cost functions are flow independent. In the deterministic case the assignment reduces to the *All or Nothing* assignment for each class, but now it means that for each OD we have to find the shortest route in $\bigcup_{m\in\mathcal{M}_k}\mathcal{R}_{od}^{k,m}$, i.e., the shortest routes considering all the possible modes.

8.1.2 Stochastic case

For the stochastic case, we can adapt the *Logit assignment* seen in the single class case. When the distribution of the error is a Gumbel distribution $\mathcal{G}\{X < x\} = e^{-e^{-\mu x}}$ the probability

p_r to choose a route $r \in R_{od}^k$ can be computed explicitly. It is given by

$$p_r = \frac{e^{-\mu C_r^{k,m}}}{\sum\limits_{r \in R_{od}^{k,m},\, m \in \mathcal{M}_k} e^{-\mu C_r^{k,m}}} , \tag{11}$$

where $C_r^{k,m}$ denotes the cost of using the route r for a class k user using mode m. We can extend the Bell formula that we obtained in the single-class case. Denoting $\mathcal{O}$ [resp. $\mathcal{D}$] the correspondent incidence node-arc-origin-class-mode [resp. node-arc-destination-class-mode] matrix we can define the *transition weight matrix* $W_{km} = \mathcal{O}\, T^{km} (\mathcal{D})'$, where T is the diagonal matrix $T_{aa}^{km} = e^{-\mu C_a^{km}}$, $a \in \mathcal{A}^{od}$,

Then the logit assignment on efficient paths is given by :

$$F_{ij} = \sum_{k, od \in \mathcal{D}^k} d_{od}^k \frac{\sum\limits_{m \in \mathcal{M}_k} (W_{km})_{oi}^* \, W_{km,ij} \, (W_{km})_{jd}^*}{\sum\limits_{m \in \mathcal{M}_k} (W_{km})_{od}^*} . \tag{12}$$

Indeed

- $(W_{km})_{od}^*$ is the total weight of all the paths from o to d that use mode m for users of the class k,
- $(W_{km})_{oi}^* \, W_{km,ij} \, (W_{km})_{jd}^*$ is the total weight of the paths that use the arc ij,
- And the quotient in (12) is the probability for a user of the class k to take a path from o to d and use the arc ij for the logit distribution on the paths.

8.2 Congested assignment

In this case we have to reconsider the Wardrop principles. Now the user equilibrium condition is: *"A user of the class k cannot change route or mode in order to reduce his travel cost"*. It means that the costs of the used routes for a given class are the same and lower or equal than the costs of the unused routes (for the same class). Mathematically it means that for each class k and each OD pair od the routes and modes can be ordered in such a way that

$$C_{r_1}^{k,m_1} = \cdots = C_{r_j}^{k,m_j} \leq C_{r_{j+1}}^{k,m_{j+1}} \leq \cdots \tag{13}$$

and

$$0 = f_{r_{j+1}}^{k,m_{j+1}} = f_{r_{j+2}}^{k,m_{j+2}} = \cdots \tag{14}$$

Which can be written equivalently, for each class k, as

$$\min\{f_r^{k,m}, C_r^{k,m} - \pi_{od}^k\} = 0 \tag{15}$$

for every $m \in \mathcal{M}_k$ and $r \in \mathcal{R}_{od}^{k,m}$, where π_{od}^k represents the minimal cost for an user of the class k going form o to d. If we incorporate the demands satisfaction conditions we obtain the mathematical formulation of the problem as

$$(15) \quad \text{and} \quad \sum_{m \in \mathcal{M}_k} (d_{od}^{k,m} - \sum_{r \in \mathcal{R}_{od}^{k,m}} f_r^{k,m}) = 0\,. \tag{16}$$

Contrary to the single class case (16) can not be interpreted as the optimality condition of an optimization problem. Moreover these variational inequalities are not, in general, monotone.

There are no known algorithms to solve non monotone variational inequalities, that is why the algorithms implemented are heuristics based on the successive averages method over the flow $F = (f_a^{k,m})$. That means that if F^n is the computed flow at the n-iteration then after recomputing the costs a new flow Y^n is obtained using one of the non-congested assignments and then the new flow is

$$F^{n+1} = \frac{n}{n+1} F^n + \frac{1}{n+1} Y^n\,. \tag{17}$$

9 Introduction to the Multiclass SCILAB Toolbox (MCIUDADSIM)

The Traffic Network toolboxes MCIUDADSIM is a SCILAB toolbox dedicated to solve the traffic assignment problem in the multiclass multimode case.

The MCIUDADSIM toolbox is available at [4]. To install MCIUDADSIM we must installed before the CIUDADSIM toolbox, see section 3. In the MCIUDADSIM toolbox the following items can be distinguished :

- Database : stored in the variable `%NET` with all the data, – geographic definition of the network, – travel demands (*commodities*), – travel times as functions of congestion (link performance functions), – the flows on the links, – the times spent on each link at equilibrium, the generalized cost of the links and the global congestion .
- Creation : functions that allow to load a previously defined example (`MTrafficExample`) and to modify it if necessary or to build up a new network.
- Edition and visualization : functions that allow to modify and graphically display the values on the graph representing the network (`MShowNet`).
- Assignment functions that apply algorithms and heuristics to compute the assignment with different criteria (`MAssign`).
- Import-export functions that allow to interface with MapInfo to facilitate the input of the data and the display of the final results (`MI2Scilab`).

10 Multiclass Database: %NET

10.1 Network Representation

A traffic network has two parts : the physical network (nodes and links) and the transport demand. The physical network is a geographic database that consists of two SCILAB typed lists the nodes (type Nodes) and the links (type Links).

The transport demand is the set of OD pairs. An OD pair consists of two nodes o and d and a real positive number that measures the number of users that go from o to d. The OD pairs will be grouped into classes, and the classes will be a set of modes the OD pairs can use. A mode will consist of a subnetwork of the original network and the list of mode-dependent parameters.

This paradigm allows to consider different values of time, different vehicles (trucks and cars), different transportation modes (bus and cars) with the possibility of sharing the demand. Therefore, in our implementation the transport demand is divided in modes that are grouped into classes

$$\text{Multi-Class Network} \begin{cases} \text{Nodes} \\ \text{Links} \\ \text{Class 1} \begin{cases} \text{Mode}_1^1 \\ \cdots \\ \text{Mode}_{m_1}^1 \end{cases} \\ \cdots \\ \text{Class k} \begin{cases} \text{Mode}_1^k \\ \cdots \\ \text{Mode}_{m_k}^k \end{cases} \end{cases}$$

and each transport demand has the structure :

$$\text{Mode}_m^k \begin{cases} \text{Available Links} \\ \text{OD pairs} \end{cases} .$$

For example, the classes of transport can be differentiated by income level (rich and poor), and each class can use two modes bus and car. In this case we have four modes:

$$\text{Rich} \begin{cases} \text{Rich People using cars} \\ \text{Rich People using bus} \end{cases} \qquad \text{Poor} \begin{cases} \text{Poor People using cars} \\ \text{Poor People using bus} \end{cases} .$$

The route costs perceived by the users are defined as the sums of the costs of the arcs belonging to the route. The cost of using the arc a by the mode m for an user of the class k is the function

$$C_a^{k,m}(f) = p_a^{k,m} + \nu^{k,m} T_a^{k,m}(f), \tag{18}$$

where $T_a^{k,m}$ is the generalized travel time, it is the sum of three quantities: the free flow travel time (t_0), a linear combination of the travel time contribution (τ) of all the classes and modes and the congestion delay (J). More precisely

$$T_a^{k,m} = t_{0a}^{k,m} + \sum_{j,\, m' \in M_j} \chi_{jm'}^{km} \tau_a^{jm'} \left(f_a^{jm'} \right) + J_a \left(\xi_a \right) \tag{19}$$

where

- $p_a^{k,m}$ is the price for the class k of using arc a traveling by mode m
- $\nu^{k,m}$ is the value of time for the class k (in general independent of the mode m),
- t_{0a}^{km} is the free flow travel time for the vehicles of the class k over the link a using the mode m, it is stored in `%NET.mode.clinks.t0`.
- τ_a^{km} is a positive, strictly increasing function that depends only on the flow f_a^{km}, of the given class over the arc a and is 0 when f_a^{km} is 0. The formula used to compute τ depends on the value of `%NET.mode_m.lpfmodel` (see table (9)), where c_a,m_a,b_a are the row vectors of the matrix `%NET.modei.mlinks.lpp`.
- ξ_a is the occupancy of the link a

  $$\xi_a = \sum_{j,n \in M_j} \xi_a^{jn} f_a^{jn},$$

  i.e., a linear combination of the flows on a.
- J_a is the delay due to the occupancy, an increasing function of ξ_a that can be chosen among

  $$\left(\frac{\xi_a}{C_a} \right)^{\beta_a} \quad \text{or} \quad \left| \xi_a - C_a \right|^{-\beta_a}$$

  where β_a is a positive real number and C_a is the practical or total capacity of the link a.

10.2 Computing the assignment:

Assignments are done by means of the function `MAssign()`. The results for each mode `modei` are stored in the variable `%NET` in the mode fields `%NET.modei.mlinks.flow`, `%NET.modei.mlinks.time`, `%NET.modei.mlinks.cost`. The global congestion results are stored in `%NET.links.occupancy` and `%NET.links.congestion`.

lpfmodel	Travel time function	Valid parameters
0	$m_a F/c_a + \max(0, m_a(F - c_a)^{b_a})$	$b_a \geq 1$, $c_a > 0$
4	$\log(c_a) - \log(c_a - F)$	$c_a > 0$
6	$c_a F + m_a F^{b_a}$	$b_a > 1$, $c_a \geq 0$

Table 9: The different formulas for the travel time computation

.

10.3 Showing the net:

When a net is shown using `MShowNet()`, three types of arcs can be seen. The black ones, light-blue ones and cyan ones. The black and light-blue arcs correspond to the roads: – the light blue to the roads with zero flow, – the black to those with a non zero flow. The cyan arcs correspond to the OD-traffic demand, these arcs go from origin to destination.

10.4 Interfacing MapInfo

In order to facilitate the data input the function `MI2Scilab` can be used. It translates a MapInfo table made under our directions (see [4]) into a SCILAB variable of type Network. More precisely, the physical network can be designed in MapInfo, then using MapInfo functions important geographic data, as links length can be calculated. After that, the network is imported in SCILAB obtaining the `%NET.nodes` and the `%NET.links` parts of `%NET`.

Once the flow is assigned (with `MAssign()`) the results can be saved as a MapInfo table using the function `Scilab2MI`. It takes the original MapInfo table and with the results in `%NET` it creates new MapInfo tables for each mode and another for the global data. These tables can be imported in MapInfo and used as different layers of the same map.

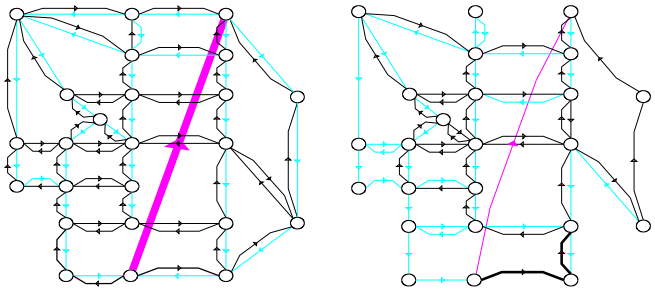
11 Multiclass Numerical Experiments

We show the results obtained over the example "Sioux Falls" where we have the class People using the modes Car and Bus, and the class Trucks with only one mode. See Figures 12 and 13.

We copy below the instructions that generated the example:

```
MTrafficExample('Sioux Falls');
MAssign();
MShowNet();
```

In order to show the kind of results we can obtain we propose the following example. Let us take a regular, ten by ten, network with two classes (rich and poor), and each class with two modes (car and bus). The following instructions generate the example data bases:



(a) Mode Car (b) Mode Bus

Figure 12: The assignment for the class People

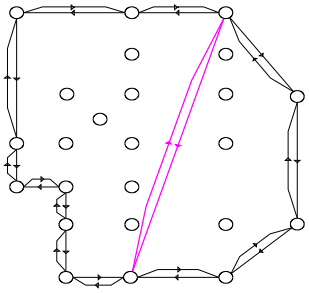


Figure 13: The assignment for the class Truck.

```
MTrafficExample('Regular2',10,'idembus',10)
%NET.properties.theta=2.5
MAssign()
%NET.properties.theta=%inf
MAssign()
```

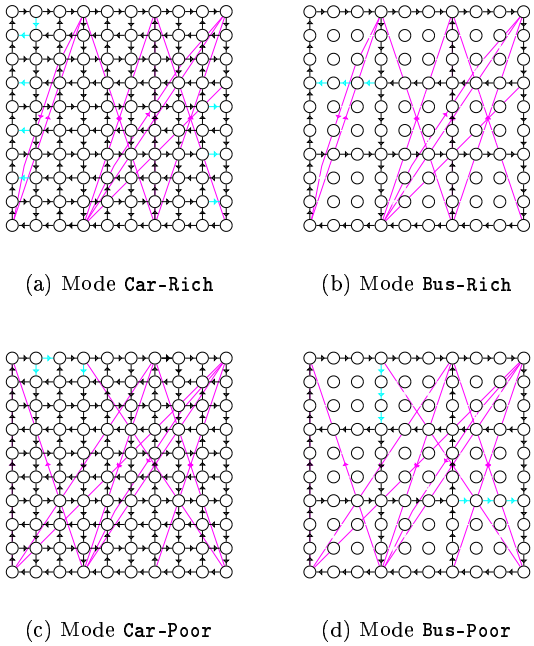
The performance results of the assignment are shown in Table 10.

In the Figure (14) we can see the assignment obtained with the successive averages method where at each iteration the assignment is made with the Logit method. In the Figure (15) we can see the assignment obtained successively averaging AON assignments.

The difference between the two assignments (Logit and AON) is approximately 2% of the total demand. Even if these results don't have to be close to each other, we can explain this fact by the size of θ which is large enough. In fact, we know, from the discussions of

Algorithm	Iter.	Time	Used Mem.	Fl. Er.u
MSA-AON	800	47.44	5520	0.0042260
MSA-Logit	200	43.29	2520	0.0000527

Table 10: Comparisons between AON and Logit for $\theta = 2.5$



(a) Mode `Car-Rich` (b) Mode `Bus-Rich`

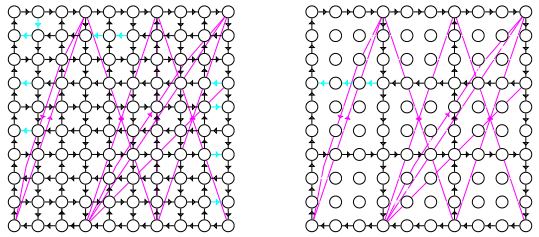
(c) Mode `Car-Poor` (d) Mode `Bus-Poor`

Figure 14: The assigned flow with successive averages of Logit assignments.

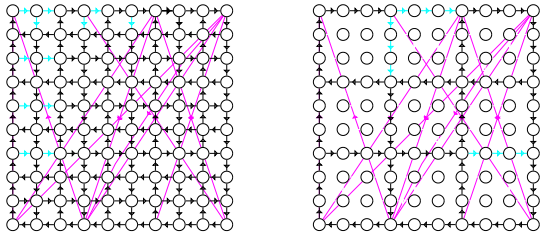
the single-class case, that for high values of θ Logit method behaves as an AON method. In the following pictures we compare both methods for the class "Poor".

We observe very different results. This behavior is due to the fact that Logit is cleverer than AON. Indeed, AON uses only one route even when there are many routes with the same cost (among the same or different modes), as it is shown in the Figure (16). Instead, the Logit method assigns the same flow to all the routes having the same lower cost, see the Figure (17).

This fact improve the speed of convergence of the computation of the logit equilibrium.

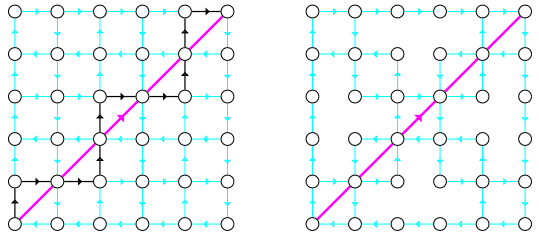


(a) Mode `Car-Rich` (b) Mode `Bus-Rich`



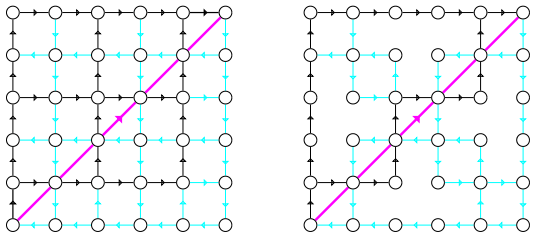
(c) Mode `Car-Poor` (d) Mode `Bus-Poor`

Figure 15: The assigned flow with successive averages of the AON algorithm.



(a) Mode `Car-Poor` (b) Mode `Bus-Poor`

Figure 16: The AON assignment for the class Poor.



(a) Mode `Car-Poor` (b) Mode `Bus-Poor`

Figure 17: The Logit assignment for the class Poor and $\theta = 2.5$.

12 Single Class Traffic Assignment Reference Manual (CiudadSim)

CiudadSim is a contribution of Scilab which is free and available at www-rocq.inria.fr/scilab/CiudadSim. It needs :

- Scilab available at
  www-rocq.inria.fr/scilab/,
- Scigraph a Scilab toolbox for visualizing graph available at
  ftp.inria.fr/INRIA/Projects/Meta2/Scilab/contrib/SCIGRAPH/
- MaxPlus a Scilab toolbox for max plus arithmetic available at
  ftp.inria.fr/INRIA/Projects/Meta2/Scilab/contrib/MAXPLUS/

We give here by alphabetic order all the function and data structures of CiudadSim. This documentation is available inline in using the help of Scilab as soon as CiudadSim has been loaded in Scilab.

It is useful also to make hierarchy in these functions. The high level ones that a user has to know to make an assignment and the low level used by the high level and useful for developers

The high level functions and data structures are

- `Netlist` contains a complete description of the structure of the database.
- `TrafficExample` to build some example of database.
- `AddLinks, AddNodes, AddDemands` to edit by Scilab programming the database.
- `TrafficAssig` to compute the assignment with various algorithms for various modelling.

- `ShowNet, ShowLinks, ShowDemands` to visualize the datas or the assignment.
- `ExportMI, MI2Scilab` to export a net to Mapinfo and to import a net from Mapinfo.

All the other functions are low level ones defining assignment algorithms, facilities or useful intermediary computations.

12.1 AON — All or Nothing traffic assignment algorithm

CALLING SEQUENCE :

```
[F]=AON(net)

[F]=AON(net,t0)

[F]=AON(tl,hl,t0,nn,origins,td,hd,dd)
```

PARAMETERS :

`net` : a NetList,
`t0` : row vector, the link travel time (if not given in net)
`tl, hl` : row vectors, tail and head nodes numbers of the links
`nn` : node number
`origins` : origins of the OD pairs
`td, hd, dd` : row vectors, tail and head nodes numbers of the demands, and demands values
`F` : assigned flow

DESCRIPTION :
Assigns the flow with the All or Nothing algorithm. For each OD-pair it looks for the minimum travel time path and assigns all the demand to this path. The travel time of the links are given by `t0` or by the corresponding field in net `net (net.links.t0)`. The non feasible OD pairs are ignored.

EXAMPLES :

```
net=TrafficExample('Small');
F=AON(net)
```

See Also : `AONd` 32, `IA` 39, `IAON` 39, `FW` 37, `CAPRES` 35, `TrafficAssig` 62

12.2 AONd — All or Nothing traffic assignment algorithm

CALLING SEQUENCE :

```
[F,f]=AONd(net)

[F,f]=AONd(net,t0)

[F,f]=AONd(tl,hl,t0,nn,origins,td,hd,dd)
```

PARAMETERS :

`net` : a NetList,
`t0` : row vector, the link travel time (if not given in net)
`tl, hl` : row vectors, tail and head nodes numbers of the links
`nn` : node number
`origins` : origins of the OD pairs
`td, hd, dd` : row vectors, tail and head nodes numbers of the demands, and demands values
`F` : assigned flow
`f` : assigned flow disaggregated by commodity

DESCRIPTION :
Displays the non feasible OD pairs, if there are some. Those pairs are ignored in the assignment. Assigns the flow with the All or Nothing algorithm. For each OD-pair it looks for the minimum travel time path and assigns all the demand to this path. The travel time of the links are given by `t0` or by the corresponding field in `net (net.links.t0)`.

EXAMPLES :

```
%net=TrafficExample('Nguyen Dupuis');
// we can see the Net
ShowNet();
//we add a non feasible OD pair
%net=AddDemands(%net,8,1,10);
ShowNet()
[F,f]=AONd(%net);
```

See Also : `AON` 31, `IA` 39, `IAON` 39, `FW` 37, `CAPRES` 35, `TrafficAssig` 62

12.3 AddDemands — Add demands to a net list

CALLING SEQUENCE :

```
AddDemands(td,hd,demand)
```

PARAMETERS :

`%net` : global Netlist variable database
`td` : row vector, tail nodes numbers of the added demands
`hd` : row vector, head nodes numbers of the added demands
`demand` : row vector, the values of the added demands

DESCRIPTION :
`AddDemands` adds demands to `%net` (the database).

EXAMPLES :

```
%net=TrafficExample('Diamond');
ShowNet()
// Add a new demand from node 3 to node 2 with value 10
//    in the  net
AddDemands(3,2,10);
// We can see the modifications with
ShowNet()
```

See Also : `IntroTrfAsg` 42, `NetList` 52, `AddLinks` 33, `AddNodes` 34, `MakeNet` 51, `ShowNet` 61

12.4 AddLinks — Add links to a net list

CALLING SEQUENCE :

```
AddLinks(tl,hl,lpp)
```

PARAMETERS :

`%net` : The global Netlist variable database
`tl` : row vector, tail nodes numbers of the added links
`hl` : row vector, head nodes numbers of the added links
`lpp` : 4 x nl matrix, travel time function parameters, where nl is the number of added links

DESCRIPTION :
`AddLinks` adds links to `%net` (the database).
EXAMPLES :

```
%net=TrafficExample('Diamond');
ShowNet()
// Add a new link from node 3 to node 4 with lpp=[0;0;1;1]
AddLinks(3,4,[0;0;1;1]);
// We can see the modifications with
ShowNet()
```

SEE ALSO : `IntroTrfAsg` 42, `NetList` 52, `AddNodes` 34, `AddDemands` 33, `MakeNet` 51, `ShowNet` 61

12.5 AddNodes — ADD NODES TO A NET LIST

CALLING SEQUENCE :

```
AddDemands(nx,ny)
```

PARAMETERS :

`net` : a Netlist
`nx` : row vector, x coordinates of the added nodes
`ny` : row vector, y coordinates of the added nodes

DESCRIPTION :
`AddNodes` adds nodes to `%net` (the database).
EXAMPLES :

```
%net=TrafficExample('Diamond');
ShowNet()
// Add 2 new nodes with coordinates x=[346,346] and y=[559,-50]
AddNodes([346,346],[559,-50]);
// We can see the modifications with
ShowNet()
```

SEE ALSO : `IntroTrfAsg` 42, `NetList` 52, `AddDemands` 33, `AddLinks` 33, `MakeNet` 51, `ShowNet` 61

12.6 CAPRES — Capres traffic assignment algorithm

CALLING SEQUENCE :

```
[f,ta]=CAPRES(net)
```

PARAMETERS :

`f` : assigned flow (disaggregated by commodity)
`ta` : times for the current assigned flow
`net` : a NetList

DESCRIPTION :
Assigns the flow with the CAPRES algorithm. As in the IAON successive AON assignments are made, but the number of iterations is fixed to 4 and instead of using the current travel times, the old travel times are combined in the same way the flows are.

EXAMPLES :

```
net=TrafficExample('Small');
[f,t]=CAPRES(net);
```

See Also : `AON` 31, `IAON` 39, `IA` 39, `FW` 37, `TrafficAssig` 62

12.7 DSD — Disaggregated Simplicial Decomposition algorithm

CALLING SEQUENCE :

```
[F,ta,ben,f]=DSD(net,routemax,nFW,eps,list_eps,list_itemax)

[F,ta,f]=dsd(net,routemax,nFW,eps,list_eps,list_itemax)
```

PARAMETERS :

`net` : a NetList
`routemax` : maximum number of general iterations
`nFW` : number of Frank-Wolfe iterations
`eps` : general precision
`list_eps` : array of precisions, Newton method precision, auxiliary precision and linear search precision

`list_itemax` : array of maximum number of iterations, Newton iterations, auxiliary iteration and linear search iteration
`F` : assigned flow
`ta` : link travel time for the assigned flow f
`ben` : `nix5` matrix (`ni` number of performed global iterations) benchmark information
`f` : assigned flow disaggregated by commodity

DESCRIPTION :
Assigns the flow with the disaggregated simplicial decomposition algorithm. This algorithm generates at each iteration a new route for each commodity using AON. The traffic is assigned minimizing the total cost over the generated routes using a Newton method. The version `DSD` shows performance statistics. The version `dsd` is used mostly inside other algorithms and it does not displays any information.

The matrix `ben` contains intermediary informations on the algorithm. For each iteration it gives the iteration number, time spent, cost, memory used and an convergence error evaluation.

EXAMPLES :

```
net=TrafficExample('Steenbrink');
// uses the example net Steenbrink
[F,ta,ben,f]=DSD(net,20,0,1e-4,[1e-6,1e-7,1e-8],[2,50,100]);
```

See Also : `AON` 31, `IAON` 39, `CAPRES` 35, `FW` 37, `TrafficAssig` 62

12.8 ExportMI — Scilab to MapInfo interface

CALLING SEQUENCE :

```
ok=ExportMI(net,name,nameout,disp_option)
```

PARAMETERS :

`net:` Scilab variable containing the network
`name:` the name of the MI source file without extension
`nameout:` the name of the MapInfo table where the results will be saved
`disp_option:` displaying options.

DESCRIPTION :
This command is meant to display in MapInfo the results obtained for a network already recovered from mapinfo. It uses an existent MI file named name and creates two files : one .mif and one.mid.

To see them in MI , they must be imported (table/import), then the values obtained in SCILAB will be in the table associated with the file, and the width of the arcs will be proportional to the chosen magnitude, flow or time.

The displaying option is used to specify the magnitude shown in the arcs. The values for `disp_option` are the strings `'flow'` and `'time'`.

EXAMPLE :

```
// 1) Choose the example Versailles.
// This network comes from  from a MapInfo file
%net=TrafficExample('Versailles');
// 2) Make an assignment
TrafficAssig()
// 3) Translate to mapinfo
ok=Scilab2MI(%net,CS_DIR+'/examples/SGL_Versailles_v',CS_DIR+..
                                        '/examples/man_example')
// 4) Look the content of the variable CS_DIR
//                             (you need it in MapInfo)
// 5) Open MapInfo and import the tables man_example,
// look for it in CS_DIR then in examples
```

SEE ALSO : `ImportMI` 40

12.9 FW — FRANK-WOLFE TRAFFIC ASSIGNMENT ALGORITHM

CALLING SEQUENCE :

```
[F,ta,ben]=FW(net,kmax,tol)
```

PARAMETERS :

`net` : a NetList
`kmax` : maximum number of iterations
`tol` : precision
`F` : assigned flow
`ta` : link travel time for the assigned flow F
`ben` : `nix5` matrix (ni number of performed iterations) benchmark information

DESCRIPTION :
Assigns the flow with the Frank-Wolfe algorithm. After an initial AON assignment the link costs are updated and a new assignment is computed. The new assignment is combined with the previous one in a way that minimizes the global cost. The algorithm terminates

when the number of iterations `kmax` is reached or the change in link costs is less than `tol`. When they are not given tol and kmax have the default values 1e-6 and 20.

The matrix `ben` contains intermediary informations on the algorithm. For each iteration it gives the iteration number, time spent, cost, memory used and an convergence error evaluation.

EXAMPLES :

```
net=TrafficExample('Small');
/// generates a network with 4 nodes, 9 arcs and 2 OD-pairs
[F,ta,ben]=FW(net);
```

SEE ALSO : `AON` 31, `IAON` 39, `CAPRES` 35, `DSD` 35, `TrafficAssig` 62

12.10 Graph2Net ——— RECOVERS THE NET FROM A GRAPH

CALLING SEQUENCE :

```
net=Graph2Net(g,nf)
```

PARAMETERS :

`g` : graph-list
`nf` : integer between 0 and 6, chooses the travel time formula
`net` : a NetList

DESCRIPTION :
Graph2Net recovers the NetList from a graph-list. In order to distinguish the demands and links arcs, the following color conventions must be satisfied in the graph:

- the links arcs must have color 1 (black),
- the demands arcs must have the color 6 (cyan).

After saving an edited traffic network (shown with `ShowNet`) in a scigraph window we get a graph-list that we can transform in a NetList with `Graph2Net`.

EXAMPLES :

```
%net=TrafficExample('Empty');
ShowNet()
// Now you can edit in the scigraph window following the
// color conventions  to  distinguish the demands  from
// the links and then save the graph.
```

```
g=load_graph('name.graph');
%net=Graph2Net(g); // By default nf=6
// You can check if the net is the same you saved.
ShowNet()
```

See Also : IntroTrfAsg 42, TrafficExample 64, ShowNet 61

12.11 IA — Incremental traffic assignment algorithm

CALLING SEQUENCE :

```
[f,ta]=IA(net,k)
```

PARAMETERS :

net : a NetList
k : number of iterations of the algorithm
f : assigned flow (disaggregated by commodity)
ta : row vector of reals, current time for the assigned flow f

DESCRIPTION :
Assigns the flow with the Incremental Assignment algorithm. Each origin-destination flow is divided into k equal parts. Each part is assigned by AON. The link costs are updated using the travel time functions of the assigned flow f.

EXAMPLES :

```
net=TrafficExample('Small');
[f,ta]=IA(net,10);
```

See Also : AON 31, IAON 39, CAPRES 35, FW 37, TrafficAssig 62

12.12 IAON — Iterated All or Nothing traffic assignment algorithm

CALLING SEQUENCE :

```
[f,ta,ben]=IAON(net,k)
```

PARAMETERS :

`net` : a NetList
`k` : number of iterations of the algorithm
`f` : assigned flow (disaggregated by commodity)
`ta` : row vector of reals, current time for the assigned flow
`ben` : a kx5 matrix benchmark information

DESCRIPTION :
Assigns the flow with the Iterated All or Nothing heuristic. Starting with a feasible flow assigned with AON the link costs are calculated with the travel time functions. The AON assignment is made now with the computed costs and then the costs are recomputed for the new assignment. This process is repeated until there is no change in the re-calculated costs or the number of iterations `k` is reached.

The matrix `ben` contains intermediary informations on the algorithm. For each iteration it gives the iteration number, time spent, cost, memory used and an convergence error evaluation.

EXAMPLES :

```
net=TrafficExample('Small');
[f,ta,ben]=IAON(net,10)
```

SEE ALSO : `AON` 31, `IA` 39, `FW` 37, `CAPRES` 35, `TrafficAssig` 62

12.13 ImportMI — MAPINFO TO SCILAB INTERFACE

CALLING SEQUENCE :

```
[net,color,Bidir]=ImportMI(name)
```

PARAMETERS :

`net` : Netlist structure : the resulting network
`color` : a vector with the color of the links
`Bidir` : a sparse matrix

DESCRIPTION :
The purpose of the MapInfo interface is to facilitate the input of the geographical data into the CIUDADSIM Toolbox and to provide a nicer way of displaying the results obtained after the assignment. There are two commands : ImportMI (to translate from MI to Scilalb) and ExportMI (to translate from SCILAB to MI).

The MapInfo files must created, for that do:

- In MapInfo, open the file MI_empty.tab located in the directory examples of the MCiudadSim toolbox.
- Save it with another name to be able to modify it.
- Close the MapInfo table MI_empty and open the one you created.
- Make the layer editable.
- Press the F key to enter in the fusion mode
- Draw the arcs as polylines.
- Export the table to obtain two files one .mif and other .mid .

Important Remarks :

1. The extreme of the arcs will be considered as nodes so if you want that two arcs share the same node is important to draw it like that. That's why the Fusion mode is important, when the pointer is over an already defined node it will change into a cross.
2. The style of the polyline is used to differentiate one-way and two-ways links. The two-ways links can have any style but one-way links must have style 2. Each two-way link in MapInfo corresponds to two one-way links in Scilab. This correspondence is found in the sparse matrix Bidir.
3. The links can be given different colors, these colors will be in the output as the vector color, it can be useful to differentiate subsets of links.

EXAMPLES :

```
// MapInfo translation from the files
//    - SGL_Versailles_v.mid
//    - SGL_Versailles_v.mif
// Regular town example with four modes and two classes.
[%net,color,Bidir]=ImportMI(CS_DIR+..
                         '/examples/SGL_Versailles_v');
// Show the translated arcs in Scilab
ShowNet()
```

See Also : ExportMI 36

12.14 IntroTrfAsg — Introduction to the Traffic Assignment Toolbox

DESCRIPTION :
The purpose of the toolbox is to solve traffic assignment problems. A good reference is Michael Patriksson's book "The Traffic Assignment Problem, Models and Methods".

To define the problem we need :

1. The Traffic Network : an oriented graph G where `nn` nodes represent places or cities, and `na` arcs represent links or roads.
2. The Link Performance Functions (also known as travel time functions): the travel time as a function of the flow in the arc. Seven formulas are used needing 4 parameters. Once a formula is chosen for the whole network, the parameters are defined in a `4×na` matrix.
3. The Origin-Destination traffic demand pairs (OD-pairs also called commodities) are given by an initial node, a final node and a flow. The `nd` OD-pairs are defined by a `3×nd` matrix.

All the data are stored in a scilab global variable `%net` which is a structure called a `NetList` that plays the role of a geographic data basis which can be visualized by `ShowNet` and edited inside "metanet" or "scigraph". This NetList is build by `MakeNet`.

The traffic assignment is computed by `TrafficAssig`. The results are stored in the Geographic Data Basis and can be visualized with `ShowNet`.

See Also : `NetList` 52, `MakeNet` 51, `ShowNet` 61, `TrafficAssig` 62

12.15 LogitB — Logit equilibrium (Bell Method)

CALLING SEQUENCE :

```
[F]=LogitB(A,D,theta)
```

PARAMETERS :

`A` : nxn nodesxnodes matrix of travel times
`D` : nxn nodesxnodes matrix of demand flows
`theta` : stochasticity parameter
`F` : nxn nodesxnodes assigned flow matrix

DESCRIPTION :
Compute the logit traffic assignment by the Bell method where all the routes are considered and not only the efficient ones (see `LogitD`).

To each route r on the graph defined by the matrix `A` is associated a weight $e^{-\theta l}$ with $\theta =$ `theta` where l denotes the the total travel time on the route r. The flow for each demand defined by the matrix `D` is assigned on a route proportionally to its weight.

The graph must be strongly connected. If it is not the case we can add artificial arcs with large travel time in such a way that the new graph becomes strongly connected.

The variable `theta` must be large enough to insure that the weight of routes with an infinite number of links are finite.

EXAMPLES :

```
// Graph generation (the graph must be stongly connected),
// n is the number of nodes, m the number of arcs.
n=10; m=40;
c1=m/(n*n);
A=sprand(n,n,c1)+diag(sparse(ones(n-1,1)),1);
A(n,1)=1;A=A-diag(diag(A));
// Demand generation, p is the number of demand.
p=30; c2=p/(n*n);D=sprand(n,n,c2);D=D-diag(diag(D));
//theta definition (almost determinsitic)
theta=10;
// Flow calculation
FD=LogitB(A,D,theta)
```

See Also : `LogitD` 43, `LogitMB` 44, `LogitMD` 45, `LogitN` 46, `LogitNE` 47, `TrafficAssig` 62

12.16 LogitD — Logit equilibrium (Dial Method)

CALLING SEQUENCE :

```
[F]=LogitD(A,D,theta)
```

PARAMETERS :

`A` : nxn nodesxnodes matrix of travel times
`D` : nxn nodesxnodes matrix of demand flows
`theta` : stochasticity parameter
`F` : nxn nodesxnodes assigned flow matrix

DESCRIPTION :
Compute the logit traffic assignment by the Dial method where efficient routes are the ones which never go back toward the origin.

To each efficient route r on the graph defined by the matrix A is associated a weight $e^{-\theta l}$ where l denotes the the total travel time on the route r. The flow for each demand defined by the matrix `D` is assigned on a route proportionally to its weight.

The graph must be strongly connected. If it is not the case we can add artificial arcs with large travel time in such a way that the new graph becomes strongly connected.

EXAMPLES :

```
// Graph generation (the graph must be stongly connected),
// n is the number of nodes, m the number of arcs.
n=10; m=40;
c1=m/(n*n);
A=sprand(n,n,c1)+diag(sparse(ones(n-1,1)),1);
A(n,1)=1;A=A-diag(diag(A));
// Demand generation, p is the number of demand.
p=30; c2=p/(n*n);D=sprand(n,n,c2);D=D-diag(diag(D));
//theta definition (almost determinsitic)
theta=10
// Flow calculation
FD=LogitD(A,D,theta)
```

See Also : `LogitB` 42, `LogitMB` 44, `LogitMD` 45, `LogitN` 46, `LogitNE` 47, `TrafficAssig` 62

12.17 LogitMB — Logit equilibrium (Markov Bell Method)

CALLING SEQUENCE :

```
[F]=LogitMB(A,D,theta)
```

PARAMETERS :

`A` : nxn nodesxnodes matrix of travel times
`D` : nxn nodesxnodes matrix of demand flows
`theta` : stochasticity parameter
`F` : nxn nodesxnodes assigned flow matrix

DESCRIPTION :
Compute the logit traffic assignment by the Markov method where on each road at each crossing we have to choose a link among the links leaving this crossing including the ones coming back to the origin.

A Markov chain is defined by normalizing the following transition matrix $W_{ij} = e^{-\theta(L_j+A_{ij}-L_i)}$ where L_j denotes the smallest travel time from j to the destination of the considered demand. The flow is assigned proportionally to the probability to use an arc before reaching the destination of the demand for the markov chain starting at the origin of the demand with a probability 1.

The graph must be strongly connected. If it is not the case we can add artificial arcs with large travel time in such a way that the new graph becomes strongly connected.

EXAMPLES :

```
// Graph generation (the graph must be stongly connected)
// n is the number of nodes, m the number of arcs
n=10; m=40;
c1=m/(n*n);
A=sprand(n,n,c1)+diag(sparse(ones(n-1,1)),1);
A(n,1)=1;A=A-diag(diag(A));
// Demand generation, p is the number of demand.
p=30;c2=p/(n*n);D=sprand(n,n,c2);D=D-diag(diag(D));
//theta definition (almost determinsitic)
theta=10
// Flow calculation
FD=LogitMB(A,D,theta)
```

See Also : LogitD 43, LogitB 42, LogitMD 45, LogitN 46, LogitNE 47, TrafficAssig 62

12.18 LogitMD — Logit equilibrium (Markov Dial Method)

CALLING SEQUENCE :

```
[F]=LogitMB(A,D,theta)
```

PARAMETERS :

`A` : nxn nodesxnodes matrix of travel times
`D` : nxn nodesxnodes matrix of demand flows
`theta` : stochasticity parameter

`F` : nxn nodesxnodes assigned flow matrix

DESCRIPTION :
Compute the logit traffic assignment by the Markov method where on each road at each crossing we have to choose a link among the efficient links leaving this crossing excluding the ones coming back to the origin.

A Markov chain is defined by normalizing the following transition matrix $W_{ij} = e^{-\theta(L_j+A_{ij}-L_i)}$ where L_j denotes the smallest travel time from j to the destination of the considered demand defined only on the efficient links. The flow is assigned

proportionally to the probability to use an arc before reaching the destination of the demand for the markov chain starting at the origine of the demand with a probability 1.

The graph must be strongly connected. If it is not the case we can add artificial arcs with large travel time in such a way that the new graph becomes strongly connected.

EXAMPLES :

```
// Graph generation (the graph must be stongly connected)
// n is the number of nodes, m the number of arcs
n=10; m=40;
c1=m/(n*n);
A=sprand(n,n,c1)+diag(sparse(ones(n-1,1)),1);
A(n,1)=1;A=A-diag(diag(A));
// Demand generation, p is the number of demand.
p=30; c2=p/(n*n);D=sprand(n,n,c2);D=D-diag(diag(D));
//theta definition (almost determinsitic)
theta=10
// Flow calculation
FD=LogitMD(A,D,theta)
```

SEE ALSO : `LogitD` 43, `LogitB` 42, `LogitMB` 44, `LogitN` 46, `LogitNE` 47, `TrafficAssig` 62

12.19 LogitN — NET LOGIT ASSIGNMENT

CALLING SEQUENCE :

```
LogitN(theta,method)
```

PARAMETERS :

`theta` : stochasticity parameter
`method` : macro among LogitB, LogitD, LogitMB, LogitMD

`%net` global variable NetList variable database

DESCRIPTION :
Compute the logit traffic assignment using a method among `LogitB, LogitD, LogitMB, LogitMD` the travel time is taken in `%net.links.time` and the assigned fow is put in the field `%net.links.flow` of the variable `%net` which is a NetList which must be declared global.
EXAMPLES :

```
// Graph generation (the graph must be stongly connected)
%net=TrafficExample('Steenbrink');
//theta definition (almost deterministic)
theta=10;
// Flow calculation
LogitN(theta,LogitMB)
ShowNet()
```

See Also : `LogitD` 43, `LogitB` 42, `LogitMB` 44, `LogitMD` 45, `LogitNE` 47, `TrafficAssig` 62

12.20 LogitNE — Net Logit Equilibrium

CALLING SEQUENCE :

```
bench=LogitNE(theta,method,eps,Niter,Num)
```

PARAMETERS :

`theta` : stochasticity parameter
`method` : macro among LogitB, LogitD, LogitMB, LogitMD
`eps` : convergence error test
`Niter` : maximal number of iteration
`Num` : number of iteration already done
`bench` : niter x 5 matrix
`%net` : global NetList variable database

DESCRIPTION :
Compute the logit traffic equilibrium using a logit method among `LogitB, LogitD, LogitMB, LogitMD` the travel time is taken in `%net.links.time` and the assigned flow is put in the field `%net.links.flow` of the variable `%net` which is a `NetList` which must be declared global.

The equilibrium is computed by a divergent series method $F_{n+1} = F_n(1-\rho_n) + \rho_n f$ where f is the new assigned flow computed with the travel time associated to the flow F_n, with $\rho_n = 1/(n+1)$.

If `%net.gp.verbose=%T` convergence informations are displayed. These informations are returned by the function in the variable bench.

The matrix `%net.gp.bench` contains intermediary informations on the algorithm. For each iteration it gives the iteration number, time spent, cost, memory used and an convergence error evaluation.

EXAMPLES :

```
// Graph generation (the graph must be stongly connected)
%net=TrafficExample('Steenbrink');
//theta definition (almost deterministic)
theta=10
// Flow calculation
LogitNE(theta,LogitMD,1.e-6,30,0)
ShowNet()
```

SEE ALSO : `LogitD` 43, `LogitB` 42, `LogitMB` 44, `LogitMD` 45, `LogitN` 46, `LogitNELS` 48, `TrafficAssig` 62

12.21 LogitNELS ——— NET LOGIT EQUILIBRIUM (LINEAR SEARCH)

CALLING SEQUENCE :

```
bench=LogitNELS(theta,method,eps,Niter)
```

PARAMETERS :

`theta` : stochasticity parameter
`method` : macro among LogitB, LogitD, LogitMB, LogitMD
`eps` : convergence error test
`Niter` : maximal number of iteration
`bench` : niter x 5 matrix
`%net` : global variable NetList
`%VERBOSE` : global boolean variable
`% STACKSIZEREF` : global integer variable

DESCRIPTION :
Compute the logit traffic equilibrium using a logit method among `LogitB, LogitD, LogitMB, LogitMD` the travel time is taken in `%net.links.time` and the assigned flow is

put in the field `%net.links.flow` of the variable `%net` which is a `NetList` which must be declared global.

The equilibrium is computed using a linear search based on the corresponding deterministic criteria as approximated Lyapounov function. This method is valid at least for large theta.

If `%net.gp.verbose=%T` convergence informations are displayed. These informations are returned by the function in the variable `%net.gp.bench`.

EXAMPLES :

```
//  Graph generation (the graph must be stongly connected)
%net=TrafficExample('Steenbrink');
//theta definition (almost deterministic)
theta=10
// Flow calculation
LogitNELS(theta,LogitMD,1.e-6,40);
ShowNet()
```

SEE ALSO : `LogitD` 43, `LogitB` 42, `LogitMB` 44, `LogitMD` 45, `LogitN` 46, `LogitNE` 47, `TrafficAssig` 62

12.22 MSA — METHOD OF SUCCESSIVE AVERAGES

CALLING SEQUENCE :

```
[f,ta,ben]=MSA(net,kmax,tol)
```

PARAMETERS :

`net` : a NetList
`kmax` : maximum number of iterations
`tol` : precision
`f` : assigned flow
`ta` : link travel time for the assigned flow `f`
`ben` : a `nix5` matrix (`ni` number of performed iterations) benchmark information

DESCRIPTION :
Assigns the flow with a MSA heuristic. After an initial `AON` assignment the links costs are updated and a new `AON` assignment is computed. The new flow, $f_{k+1} = (1 - \rho_k) f_k + \rho_k y_k$, where $\rho_k = 1/k$, is computed as a combination of the previous flow f_k and an `AON` assignment y_k. This is repeated iteratively until iteration `kmax` or until the precision `tol` is reached.

The matrix ben contains intermediary informations on the algorithm. For each iteration it gives the iteration number, time spent, cost, memory used and an convergence error evaluation.

EXAMPLES :

```
net=TrafficExample('Steenbrink');
[f,s,ben]=MSA(net,15,1e-3);
```

SEE ALSO : AON 31, IAON 39, CAPRES 35, FW 37, TrafficAssig 62

12.23 MSASUE — MSA ALGORITHM FOR STOCHASTIC USER EQUILIBRIUM

CALLING SEQUENCE :

```
[f,ta,ben]=MSASUE(net,beta,kmax,tol)
```

PARAMETERS :

net : a NetList
beta : variance of the perceived travel time over a road segment of unit travel time
kmax : maximum number of iterations
tol : precision
f : assigned flow (disaggregated by commodity)
ta : link travel time for the assigned flow f
ben : a nix5 matrix (ni number of performed iterations) benchmark information

DESCRIPTION :

Assigns the flow with a MSA heuristic using a Probit-based model in each iteration. After an initial Probit assignment the links costs are updated and a new Probit assignment is computed using flow dependent travel times. The new flow, $f_{k+1} = (1 - 1/k)f_k + (1/k)y_k$, is computed as a combination of the previous flow f_k and a Probit assignment y_k. This is repeated iteratively until iteration kmax or until the precision tol is reached.

The matrix ben contains intermediary informations on the algorithm. For each iteration it gives the iteration number, time spent, cost, memory used and an convergence error evaluation.

EXAMPLES :

```
net=TrafficExample('Small');
[f,s,ben]=MSASUE(net,1,15,1e-3);
```

SEE ALSO : Probit 56, MSA49, FW 37, DSD 35, TrafficAssig 62

12.24 MakeNet — MAKES A NET LIST

CALLING SEQUENCE :

```
net=MakeNet(nn,nx,ny,tl,hl,nf,lpp,td,hd,demand)
```

PARAMETERS :

`nn` : integer, the number of nodes in the net
`nx` : row vector, x coordinates of the nodes
`ny` : row vector, y coordinates of the nodes
`tl` : row vector, tail nodes numbers of the links
`hl` : row vector, head nodes numbers of the links
`nf` : integer between 0 and 6, travel time function formula
`lpp` : $4 \times n_l$ matrix, travel time function parameters, where n_l is the number of links
`td` : row vector, tail nodes numbers of the demands (its size is the number of commodities)
`hd` : row vector, head nodes numbers of the demands
`demand` : row vector, demand flows
`net` : a NetList

DESCRIPTION :
`MakeNet` makes a net list according to its arguments.

EXAMPLES :

```
// Coordinates for the nodes
nx=[500 10 500 900]
ny=[10 300 600 300]
// tail and head vectors for the links
tl=[1 2 3 1 1 3]
hl=[2 3 4 4 3 1]
// link-performance-function parameters
lpp=[1 1 1 1 1 1;1 3 5 2 7 1;1 2 1 2 1 2;2 2 2 2 2 2];
// Creation of the traffic net
%net=MakeNet(4,nx,ny,tl,hl,6,lpp,1,3,10);
// We can see it with
ShowNet()
```

SEE ALSO : `NetList` 52, `ShowNet` 61

12.25 Net2Par — PARAMETERS FROM NET

CALLING SEQUENCE :

```
[g,d,lpp,nf]=Net2Par(net)
```

PARAMETERS :

`net` : a NetList
`g` : graph-list, graph of the network
`d` : 3 x nd matrix, tail, head and value of the demands
`lpp` : 4 x na matrix, data for the link performance functions
`nf` : integer between 0 and 6, travel time function formula

DESCRIPTION :
`Net2Par` recovers the data fields from a NetList.

EXAMPLES :

```
net=TrafficExample('Sioux Falls');
[g,d,lpp,nf]=Net2Par(net);
```

SEE ALSO : `IntroTrfAsg` 42, `NetList` 52, `Par2Net` 55, `TrafficExample` 64

12.26 NetList — TRAFFIC ASSIGNMENT GEOGRAPHIC DATA BASE

NETLIST :
All the data needed and obtained in a traffic assignment problem are stored in a SCILAB data structure that we call a NetList which is a typed list defined by :

```
tlist(['net','gp','nodes','links','demands'],gp,nodes,links,..
                                                    demands)
```

where :

- `gp` : contains the general properties of the net,
- `nodes` : contains the nodes parameters
- `links` : contains the links parameters
- `demands` : contains the demands parameters

Each one of those elements is itself a typed list.
GP :

- `node_number` : number of nodes in the network
- `link_number` : number of links in the network
- `demand_number` : number of demands (commodities)
- `lpf_model` : integer between 0 and 6 which defines the form of the link performance function.
- `verbose` (default `%T`): boolean when it is true information are given during assignment algorithm
- `algorithm` (default `'DSD'`): function name defining the algorithm chosen for the assignment
- `NodeDiameter` (default 30) : integer used in the graph window for displaying the network
- `NodeBorder` (default 1): integer giving the width of the circle border of the node symbols
- `FontSize` (default 10): integer used in the graph window
- `tolerance` (default 1e-6): real giving the precision used in algorithms
- `theta` (default 1): stochasticity parameter for Logit
- `Niter` (default 200): integer giving maximal number of iteration
- `N0` (default 0): integer giving starting number for Mean Average Algorithm
- `ShowDemands` (default `%T`): boolean when it is true the demands are shown by `ShowNet`
- `Show` (default `'flow'`): one of the string `'flow' 'cost' or 'time'`; it is used in `ShowNet` to choose the shown link weights
- `bench` (default 0): is an array storing the intermediary results computed by the algorithm when `gp.verbose=%T` this information is shown also during the computation.

NODES :

- `name` : string row of node names
- `x` : real row of the node x-coordinates
- `y` : real row of the node y-coordinates

LINKS :

- `name` : string row vector of link names
- `tail` : integer row vector of tail node numbers of the links
- `head` : integer row vector of head node numbers of the links
- `lpf_data` : array of 4 rows vectors giving the parameters of the lpf function of the links
- `flow` : real row vector of the total flows of the links (0 by default)
- `time` : real row vector of the time spent in the links
- `disaggregated_flow` : nd x na real matrix of the commodity flows on the links

DEMANDS :

- `name` : string row vector of demand names
- `tail` : integer row vector of tail node numbers
- `head` : integer row vector of head node numbers
- `demand` : real row vector of demand flows

EXTRACTING AND INSERTING DATA FROM A NETLIST :
As they are typed lists it is always possible to access to any field using the access functions, for example the x-coordinates of the nodes is `net1.nodes.x` .

LINK PERFORMANCE FUNCTION :
It is possible to choose among different formulas for the link performance functions with the value of `lpf_formula` field of `gp` denoted here `nf`.

$$
\begin{aligned}
\texttt{nf} = 0: &\quad c = t_0 + m_a/(c_a F) + m_a[\max(0, F - c_a)]^{b_a}, \text{ with } b_a \geq 1 \\
\texttt{nf} = 1: &\quad c = t_0 e^{(F/ca)-1}, \text{ with } c_a > 0 \\
\texttt{nf} = 2: &\quad c = t_0 2^{(F/ca)-1}, \text{ with } c_a > 0 \\
\texttt{nf} = 3: &\quad c = t_0[1 + 0.15(F/c_a)^{m_a}], \text{ with } m_a \geq 1 \text{ and } c_a > 0 \\
\texttt{nf} = 4: &\quad c = t_0 + \log(c_a) - \log(c_a - F), \text{ with } c_a > 0 \\
\texttt{nf} = 5: &\quad c = b_a - c_a(t_0 - b_a)/(F - c_a), \text{ with } c_a > 0 \\
\texttt{nf} = 6: &\quad c = t_0 + c_a F + m_a F^{b_a}, \text{ with } b_a > 1
\end{aligned}
$$

where the t_0, c_a, m_a, b_a are the row vectors of the matrix `net.links.lpf_data`

SHOWING THE NET :
When a net is shown, three types of arcs can be seen. The black, light-blue and cyan.

1. The black and light-blue arcs correspond to the roads.
   - The light blue to those roads with zero flow.
   - The black to those with a non zero flow.
2. The cyan arcs correspond to the OD-traffic demand. These arcs go from origin to destination.

SEE ALSO : IntroTrfAsg 42, MakeNet 51, ShowNet 61

12.27 Par2Net — NET FROM PARAMETERS

CALLING SEQUENCE :

```
net=Par2Net(g,d,lpp,nf)
```

PARAMETERS :

net : a NetList
g : graph-list, graph of the network
d : 3xnd matrix, tail, head and value of demands
lpp : 4xna matrix, data for the link performance functions
nf : integer between 0 and 6, travel time function formula

DESCRIPTION :
Par2Net Assembles the data fields in a net.

EXAMPLES :

```
net=TrafficExample('Sioux Falls');
[g,d,lpp,nf]=Net2Par(net);
net2=Par2Net(g,d,lpp,nf)
```

SEE ALSO : IntroTrfAsg 42, NetList 52, Net2Par 52, TrafficExample 64

12.28 Probit — PROBIT-BASED STOCHASTIC NETWORK ASSIGNMENT

CALLING SEQUENCE :

```
[f,s,ben]=Probit(net,ta,beta,accuracy,kmax)
```

PARAMETERS :

`net` : a NetList
`ta` : link travel time
`beta` : variance of the perceived travel time over a road segment of unit travel time
`accuracy` : precision
`kmax` : maximum number of iterations
`f` : assigned flow (disaggregated by commodity)
`s` : standard deviation for the assigned flow f

DESCRIPTION :
Assigns the flow with a Probit-based model. In this model the perceived travel time along any given path is normally distributed with mean `ta` and variance `beta*ta`. The algorithm use to make the assignment is base on a Monte Carlo simulation of the perceived link travel times.

EXAMPLES :

```
net=TrafficExample('Small');
ta=net.links.lpf_data(1,:);
// uses the example net Small
[f,s,ben]=Probit(net,ta,0.1,0.001,7);
```

SEE ALSO : `AON` 31, `IAON` 39, `CAPRES` 35, `FW` 37, `TrafficAssig` 62

12.29 RandomNNet _ RANDOM GENERATION OF TRAFFIC NETWORK DATA

CALLING SEQUENCE :

```
net=RandomNNet(nn,mna,var,nd,nf)
```

PARAMETERS :

`nn` : number of nodes

`mna` : mean value of the exterior degree of each node (numbers of leaving arcs)
`var` : variance of the exterior degree of each node
`nd` : number of OD-pairs (origin-destination)
`nf` : model chosen for the computation of the travel time
`net` : resulting network

DESCRIPTION :
Generates randomly the data for the Traffic Assignment Problem. For each node an exterior degree is chosen randomly with a normal distribution of mean `mna` and variance `var`, then the successors are chosen without repetition. The value of `nf` indicates the formula chosen for the computation of the travel time, by default it is 6. This way of generating randomly the net if very much slower than `RandomNet`.
EXAMPLES :

```
%net=RandomNNet(3,3,1,2);
// generates a network with 3 nodes,2 OD-pairs and form
// each node the number of leaving arcs is normally dis-
// tributed with mean 3 and var 1
ShowNet();
```

See Also : `IntroTrfAsg` 42, `RandomNet` 57, `Regular` 58, `NetList` 52, `ShowNet` 61

12.30 RandomNet — Random generation of traffic network data

CALLING SEQUENCE :

```
net=RandomNet(nn,na,nd,nf)
```

PARAMETERS :

`nn` : number of nodes
`na` : number of arcs
`nd` : number of OD-pairs (origin-destination)
`nf` : model chosen for the computation of the travel time
`net` : the resulting Netlist

DESCRIPTION :
Generates randomly a strongly connected network with `nn` nodes, (approximately) `na` links and (approximately) `nd`Ô OD pairs. The value of `nf` indicates the formula chosen for the computation of the travel time, by default it is 6.
EXAMPLES :

```
%net=RandomNet(3,5,2);
// generates a network with 3 nodes, 5 arcs and 2 OD-pairs
ShowNet();
```

SEE ALSO : IntroTrfAsg 42, RandomNNet 56, Regular 58, ShowNet 61

12.31 Regular _ GENERATION OF REGULAR CITY TRAFFIC NETWORK DATA

CALLING SEQUENCE :

```
net=Regular(hs,vs,nd)
```

PARAMETERS :

hs : integer number of horizontal streets
vs : integer number of vertical streets
nd : non-negative real parameter related to the OD-pairs (origin-destination)
net : the resulting Netlist

DESCRIPTION :
Generates a NetList consisting in a regular city whose graph is a hs×vs grid For the link travel time functions, the data is the same for all links and given by t0=ca=ma, ba=1. The lpf_model is 3. It is possible to generated 4 different types of Regular City depending on parameter nd:

- nd=[] : The net has all possible O-D pairs all of them with demand equal to 1
- nd=1 : There is only one O-D pair joining the most distant nodes. Demand value equal to 1.
- nd $\in (0,1)$: The parameter nd represent the density of O-D pairs present in the net. Demands value equal to 1.
- nd>1 : There are nd random demands generated. Demands value in the interval $[0, 15]$.

EXAMPLES :

```
%net=Regular(4,5);
// generates a network with 4 horizontal streets
//  and 5 vertical streets,
// (20 nodes, 31 arcs) and all possible OD-pairs (343)
```

```
ShowNet();
%net=Regular(4,5,1);
// generates only one demand
ShowNet();
%net=Regular(4,5,0.5);
// generates randomly the OD-pairs with density 0.5
ShowNet();
%net=Regular(4,5,7);
// generates 7 random OD-pairs
ShowNet();
```

SEE ALSO : IntroTrfAsg 42, TrafficExample 64, ShowNet 61

12.32 ShowDemands ——— SHOWS THE DEMANDS OF A NET USING METANET OR SCIGRAPH

CALLING SEQUENCE :

```
ShowDemands()
```

```
ShowDemands(nodes,demands,dmin,dmax)
```

PARAMETERS :

%net : the global variable NetList database
nodes : string or row vector of integers (nodes)
demands : string or row vector of integers (demands)
dmin,dmax : reals, minimum and maximum demands

DESCRIPTION :
ShowDemands shows the demands of %net that have volumes between dmin and dmax. All the demands are shown if nothing is indicated)

EXAMPLES :

```
%net=TrafficExample('Steenbrink');
ShowDemands() // Shows the net
ShowDemands('all','between',100,200)
ShowDemands('used','between',100,200)
// Shows the net with the demands between 100 and 200,
// and the used nodes.
ShowDemands('used',[1 3 8 12],100,200)
// Shows the given demands with values between 100 and 200
```

SEE ALSO : ShowNet 61, ShowLinks 60, TrafficExample 64, TrafficAssig 62, IntroTrfAsg 42, NetList 52

12.33 ShowLinks _ SHOWS THE LINKS OF A NET USING METANET OR SCIGRAPH

CALLING SEQUENCE :

```
ShowLinks()
ShowLinks(nodes,arcs,fmin,fmax,tmin,tamx)
ShowLinks(nodes,arcs,fmin,fmax,tmin,tamx,d)
```

PARAMETERS :

%net : the global NetList variable database
nodes : string or row vector of integers (nodes)
arcs : string or row vector of integers (arcs)
fmin,fmax : reals, minimum and maximum flow
tmin,tmax : reals, minimum and maximum time
d : integer, a specific demand

DESCRIPTION :
Similar to ShowNet but it doesn't show the demands, it only shows the links of %net that verify the conditions given by the parameters (see ShowNet). If a demand is indicated, then the flow shown is the corresponding to that demand.

EXAMPLES :

```
%net=TrafficExample('Steenbrink');
%net.gp.algorithm='AON';
TrafficAssig();
ShowLinks() // Shows the net
ShowLinks('used','between',3000,6000,0,0,3)
// Shows the net with the flow corresponding to demand 3,
// the light blue arcs are not used by the commodity 3
// but by other commodities. The width corresponds to
// the flow of the commodity 3. The arcs not shown have
// total flow less than 3000 or greater than 6000.
```

SEE ALSO : ShowNet 61, ShowDemands 59, TrafficAssig 62, IntroTrfAsg 42, NetList 52

12.34 ShowNet — Show a net using Metanet or Scigraph

CALLING SEQUENCE :

```
ShowNet()
ShowNet(nodes,arcs,fmin,fmax,tmin,tamx,demands,dmin,dmax)
ShowNet(nodes,arcs,fmin,fmax,tmin,tamx,demands,dmin,dmax,d)
```

PARAMETERS :

`%net` : the global NetList varaible database
`nodes` : string or row vector of integers (nodes)
`arcs` : string or row vector of integers (arcs)
`fmin,fmax` : reals, minimum and maximum flow
`tmin,tmax` : reals, minimum and maximum time
`demands` : string or row vector of integers (demands)
`dmin,dmax` : reals, minimum and maximum demand
`d` : integer, a demand number

DESCRIPTION :
`ShowNet` shows the state of the `%net` with the flow and the travel time if they are already assigned. The conventions are that the non used arcs are light blue, the used arcs are black with their width proportional to the flow. The demands are magenta with their width proportional to the demand flow. The orientation of the demand arcs is from origin to destination.

It is possible to give a list of nodes, arcs and demands to be shown or a predefined string

- For the nodes the strings available are : `'all'` representing all the nodes, and `'used'` representing the nodes used by some route.
- For the arcs the options are `'all'` or `'between'` in this case the flow limits are the parameters `fmin, fmax` and the travel time limits are `tmin, tmax`.
- For the demands the options are `'all'` or `'between'` in this case the demand limits are `dmin,dmax`.

It is also possible to choose a demand in order to show the disaggregated flow corresponding to a specific commodity. In this case the limits on the flow given by fmin and fmax corresponds to the total flow but the width and the colors of the arcs correspond to the disaggregated flow.

EXAMPLES :

```
%net=TrafficExample('Steenbrink');
%net.gp.algorithm='DSDisaggregated';
TrafficAssig();
ShowNet() // Shows the net
ShowNet('used','between',1000,2000,0,0,3,0,10000,3)
// Shows the net with the flow corresponding to demand 3,
// the light blue arcs are not used by the commodity 3
// but by other commodities. The width corresponds to
// the flow of the commodity 3. The arcs not shown have
// total flow less than 1000 or greater than 2000.
```

SEE ALSO : ShowLinks 60, ShowDemands 59, TrafficExample 64, TrafficAssig 62, IntroTrfAsg 42, NetList 52

12.35 TrafficAssig — TRAFFIC ASSIGNMENT

CALLING SEQUENCE :

```
TrafficAssig(p1,p2,p3)
```

PARAMETERS :

%net : : global NetList variable containing the Net database
%net.gp.algorithm : string to choose among 'AON', 'CAPRES', 'DSD', 'FW', 'IA', 'IAON', 'MSA', 'Probit', 'ProbitE', 'LogitB', 'LogitD', 'LogitMB', 'LogitMD', 'LogitBE', 'LogitDE', 'LogitMBE', 'LogitMDE'
p1,p2 p3 : different algorithms' parameters to change the default ones given in fields of %net.gp (see NetList).

DESCRIPTION :
Assigns the flow of the traffic network described by the NetList net using the algorithm selected by algo with the parameters given by p1, p2 and p3 (default in %net.gp) :

- 'AON' : All or nothing
- 'CAPRES' : Capres algorithm
- 'DSD' : Disaggregated simplicial decomposition (only total flow iscomputed). p1: iterations number, p2: precision
- 'DSDisaggregated' : Disaggregated simplicial decomposition. p1: iterations number, p2: precision

- `'FW'` : Frank-Wolfe algorithm. p1: iterations number, p2: precision
- `'IA'` : Incremental assignment. p1: iterations number
- `'IAON'` : Iterated all or nothing. p1: iterations number
- `'MSA'` : Method of successive averages. p1: iterations number, p2: precision
- `'Probit'` : Probit assignment. p1: beta, p2: precision, p3: iterations number
- `'ProbitE'` : MSA algorithm for the stochastic user equilibrium (MSASUE). p1:beta, p2: precision, p3: iterations number
- `'LogitB'` : Logit assignment (all paths used not only efficient ones used)
- `'LogitD'` : Logit assignment (only efficient paths are used)
- `'LogitMB'` : Reiterated first step logit assignment (all paths used not only efficient ones used)
- `'LogitMD'` : Reiterated first step logit assignment (only efficient paths are used)
- `'LogitBE'` : Logit equilibrium assignment (all paths used not only efficient ones used)
- `'LogitDE'` : Logit equilibrium assignment (only efficient paths are used)
- `'LogitMBE'` : Reiterated first step logit equilibrium assignment (all paths used not only efficient ones used)
- `'LogitMDE'` : Reiterated first step logit equilibrium assignment (only efficient paths are used)
- `'NwtArc'` : Newton method based on nodes-links formulation (useful

when there are a small number of demands)

The assigned flow, the current travel time and the disaggregated flow are stored in `%net`.

It is correct that for some parameters and network some assignment methods does not succeed to give an assignment. In particular, for Logit method, the effective domain of the stochasticity parameter is limited and depends of the network. For some networks and some methods this domain may be empty (for example LogitB with a zero travel time loop). For Logit method, the stochasticity parameter must not be too large to avoid numerical difficulties coming from the limited precision of the computation.

EXAMPLES :

```
//
// To load an editable traffic network graph
//
%net=TrafficExample('Steenbrink');
ShowNet();
//
// The black arcs are the roads and the cyan ones are the
// traffic OD demands. We can edit the net (see the net-edition demo).
//
// To compute the traffic assignment with 'DSD' algorithm
//
TrafficAssig();
ShowNet();
//
// To show  the travel times select :
//       GRAPH/Display arc names/Time
//
// It is possible to assign the flow using another algorithm
// for example the incremental assignment denoted  'IA'
//
%net.gp.algorithm='IA'
%net.gp.Niter=50;
TrafficAssig();
ShowNet();
//
// For Logit assignment, the stochasticity parameter
// is in the field %net.gp.theta (default 1)
//
%net.gp.algorithm='LogitMD';
%net.gp.theta=3;
TrafficAssig();
ShowNet();
```

See Also : AON 31, CAPRES 35, DSD 35, FW 37, IA 39, IAON 39, MSA 49, Probit 56, LogitB 42, LogitD 43, LogitMB 44, LogitMD 45, LogitN 46, LogitNE 47, LogitNELS 48, MSASUE 50, IntroTrfAsg 42, NetList 52

12.36 TrafficExample — Traffic network example

CALLING SEQUENCE :

```
net=TrafficExample('name',p1,p2,p3,p4)
```

PARAMETERS :

`net` : NetList of the traffic network
`name` : string to choose among `'Braess'`, `'Chicago'`, `'Diamond'`, `'Empty'`, `'Nguyen Dupuis'`, `'Normal Random City'`, `'Random City'`, `'Regular City'`, `'Sioux Falls'`, `'Small'`, `'Steenbrink'`, `'Triangle'`
`p1, p2, p3, p4` : reals needed for examples for `'Regular City'`, `'Random City'` and `'Normal Random City'`

DESCRIPTION :
Generates a net-list choosing among many traffic network examples:

- `'Braess'` : 4 nodes, 5 links and 1 demand. Linear travel time functions.
- `'Chicago'` : 546 nodes, 2176 links and 93135 demands. Real large example.
- `'Diamond'` : 8 nodes, 14 links and 1 demand, diamond shaped network.
- `'Empty'` : 2 nodes and 1 link graph.
- `'Nguyen Dupuis'` : 13 nodes, 19 links and 4 demands network. Linear travel time functions.
- `'Normal Random City'` : generated using `RandomNNet(p1,p2,p3,p4)`.
- `'Random City'` : generated using `RandomNet(p1,p2,p3,p4)`.
- `'Regular City'` : generated using `Regular(p1,p2,p3)`.
- `'Sioux Falls'` : 24 nodes, 76 links and 528 demands network. Travel time functions of power 4.
- `'Small'` : 4 nodes, 9 links and 2 demands network. Quadratic travel time functions.
- `'Steenbrink'` : 9 nodes, 36 links and 12 demands network. Linear travel time functions.
- `'Triangle'` : 3 nodes and 6 links network

These examples can be used as starting point to make different networks.

EXAMPLES :

```
%net=TrafficExample('Sioux Falls');
ShowNet()
```

SEE ALSO : `RandomNet` 57, `RandomNNet` 56, `Regular` 58, `NetList` 52, `TrafficAssig` 62

12.37 WardropN ——— WARDROP EQUILIBRIUM (NEWTON HYBRID METHOD)

CALLING SEQUENCE :

```
bench=WardropN(a,k,eps,niter)
```

PARAMETERS :

`a` : regularizing coefficient (small number)
`k` : reducing regularizing coefficient exponential rate (number < 1)
`eps` : precision of the convergence test (small number)
`niter` : maximal number of iterations
`bench` : niter x 5 matrix
`%net` : global variable NetList

DESCRIPTION :
Compute the Wardrop equilibrium of a transport assignment problem by solving the following nodes-links variational formulation of the problem :

$$\min_q \sum_l C_l(f_l), \quad f_l = \sum_i q_{li}, \quad Hq_i = d_i, \quad 0 \leq q_{li}.$$

where :

- $C(f)$ is a cost in the classes defined by the lpf function,
- H is is the incidence nodes-arcs matrix of the network,
- q_{li} is the flow of the commodity i on the link l,

`%net` is the netlist containing all the information relative to the network and the cost function used.

The method used is a decomposed newton method in the space (q_i, v_i) $i = 1, p$, where the v_i denote the dual variables associated to the constraints $Hq_i = d_i$.

The variable a regularizes the matrices giving the potentials. At each iteration a is reduced by a factor k. We can try first a small a and $k = 1$.

This method is useful mainly in the case of a small number of commodities.

The matrix `bench` contains intermediary informations on the algorithm. For each iteration it gives the iteration number, time spent, cost, memory used and an convergence error evaluation.

EXAMPLES :

```
// Definition of the Network
nw=4;
%net=TrafficExample('Regular City',nw,nw);
// Traffic Assignment
bench=WardropN(0.1,2,1.e-4,12);
// Visualization of the NET
ShowNet();
```

See Also : DSD 35, FW 37, Probit 56, MSASUE 50, LogitNE 47, LogitNELS 48, TrafficAssig 62

12.38 dlpf — Plot the travel time functions

CALLING SEQUENCE :

```
c=dlpf(f0,inc,f1,lpp,nf)
```

PARAMETERS :

f0 : minimal flow
inc : increment
f1 : maximal flow
lpp : matrix of travel time function parameters
nf : ttf formula
c : matrix of the computed costs

DESCRIPTION :
Plot the travel time functions for the given parameters (lpp) between the minimal flow f0 and the maximal flow f1 using a step of inc and the formula nf

EXAMPLES :

```
net=TrafficExample('Small');
[g,d,lpp]=Net2Par(net);
// To show the function with the different parameters
dlpf(0.1,.01,2,lpp,6);
```

See Also : IntroTrfAsg 42, TrafficExample 64, lpf 68

12.39 lpf — Travel time functions

CALLING SEQUENCE :

```
[ta,dta,cta]=lpf(F,lpp,nf)
```

PARAMETERS :

`F` : row vector of flows in each arc
`lpp` : parameters of the travel time functions (ttf) for each arc
`nf` : model for the ttf
`ta` : travel time of each arc for the given flow
`dta` : derivate of the ttf for the given arc flow
`cta` : integral of the ttf for the given arc flow

DESCRIPTION :
Computes the travel time of each arc for the given flow using the travel time functions with parameters `lpp` and formula `nf`. Each column of `lpp` is of the form `[t0; ca; ma; ba]` where the coefficients `t0` is the free-flow travel time and `ca` the practical capacity of the link. Associated to the `nf` number correspond a formula giving the travel time see `NetList` for the precise definition of these formulas.

EXAMPLES :

```
net=TrafficExample('Small');
[g,d,lpp,nf]=Net2Par(net);
F=rand(1,9);
[ta,dta,cta]=lpf(F,lpp,nf)
// shows the function with the different parameters
dlpf(0.1,.01,5,lpp,6);
```

See Also : `IntroTrfAsg` 42, `TrafficExample` 64, `dlpf` 67

13 Multiclass Traffic Assignment Reference Manual (MCiudadSim)

The MCiudadSim (MCS) toolbox is available at
`http://www-rocq.inria.fr/scilab/CiudadSim`. MCS has been developed in *C++* and Scilab script code. Scilab and a few toolboxes must be installed before installing MCS :

- Scilab available at
  `http://www-rocq.inria.fr/scilab/`,

- SCIGRAPH a SCILAB toolbox for visualizing graph available at
  `ftp.inria.fr/INRIA/Projects/Meta2/Scilab/contrib/SCIGRAPH/`
- MAXPLUS a SCILAB toolbox for max plus arithmetic available at
  `ftp.inria.fr/INRIA/Projects/Meta2/Scilab/contrib/MAXPLUS/`
- CIUDADSIM the single class version available at
  `http://www-rocq.inria.fr/scilab/CiudadSim`

We give here, in alphabetic order, all the function and data structures of MCIUDADSIM. This documentation is available on-line using the help of SCILAB after the loading of the toolbox.

It is useful to make a hierarchy in these functions. The highest level corresponds to the functions that a standard user should know to make an assignment and the lowest level functions are used by the higher level ones and are useful for developers.

In order to allow these functions to coexist with the ones in the CIUDADSIM toolbox we adopted the convention of begin the function names with a capital M.

The highest level functions and data structures are

- `MNetlist` contains a complete description of the structure of the database.
- `MTrafficExample` to build some example of database.
- `AddLinks, AddNodes, AddDemands` to edit by SCILAB programming the database.
- `MTrafficAssig` to compute the assignment with various algorithms for various modeling.
- `MShowNet, MShowLinks, MShowDemands` to visualize the data or the assignment.
- `Scilab2MI, MI2Scilab` to export a net to MapInfo and to import a net from MapInfo.

All the other functions are lower level defining assignment algorithms, facilities or useful intermediary computations.

13.1 MAON — MULTI-MODE ALL OR NOTHING TRAFFIC ASSIGNMENT ALGORITHM

CALLING SEQUENCE :

```
MAON()
```

DESCRIPTION :
Assigns the flow with the All or Nothing algorithm in the multi-mode case. For each OD-pair and each mode it looks for the minimum cost path and mode and assigns all the demand to this path and mode. The cost of the links of the mode `modei` is taken from `%NET.modei.mlinks.cost` and the assigned flow is written in `%NET.modei.mlinks.flow`
EXAMPLES :

```
%NET=MTrafficExample('regular',4,1,1);
MAON()
```

SEE ALSO : `MLogitB` 72, `MLogitN` 73, `MLogitNE` 70, `Network` 76

13.2 MAssign — MULTI MODE LOGIT EQUILIBRIUM

CALLING SEQUENCE :

```
MAssign() or MlogitNE()
```

DESCRIPTION :
Computes the multi-mode logit traffic equilibrium (with stochasticity parameter given in the field `%NET.properties.theta`) using successive averages of logit assignments (when `theta=%inf`, instead of the logit assignment, the All or Nothing assignment is used) the travel cost of each mode is taken in `%NET.modei.mlinks.cost` and the assigned flow is put in the field `%NET.modei.mlinks.flow` of the variable `%NET`.

The successive averages method means that the equilibrium is computed by a divergent series method, i.e., let $\rho_n = 1/(n+1)$ we define

$$F_{n+1} = F_n(1-\rho_n) + \rho_n f_n\,,$$

where f_n is the flow assigned recomputing the travel times for the flow F_n.

If `%NET.properties.verbose =%T` information about the convergence of the algorithm is displayed. This information is also returned in the field `%NET.properties.bench`.

EXAMPLES :

```
// Graph generation (the graph must be stongly connected)
// regular town example with four modes and two classes
%NET=MTrafficExample('small');
//theta definition
%NET.properties.Niter=20;
%NET.properties.theta=1.5;
```

```
// Flow calculation
MAssign()
MShowNet()
```

SEE ALSO : `MLogitB` 72, `MLogitN` 73, `MAON` 69

13.3 MI2Scilab — MAPINFO TO SCILAB INTERFACE

CALLING SEQUENCE :

```
[nodes,links,color,Bidir]=MI2Scilab(name)
```

PARAMETERS :

`nodes` the list of nodes of the network
`links` the list of links of the network
`color` a vector with the color of the links
`Bidir` a boolean vector

DESCRIPTION :
The purpose of the MAPINFO interface is to facilitate the input of the geographical data into the CIUDADSIM Toolbox and to provide a nicer way of displaying the results obtained after the assignment. There are two commands : `MI2Scilab` (to translate from MI to SCILAB) and `Scilab2MI` (to translate from SCILAB to MI).

The MapInfo files must be created following these directions:

- In MapInfo, open the file `MI_empty.tab` located in the directory examples of the MCIUDADSIM toolbox.
- Save it with another name to be able to modify it.
- Close the MapInfo table `MI_empty` and open the one you created.
- Make the layer editable.
- Press the F key to enter in the fusion mode
- Draw the arcs as polylines.
- Export the table to obtain two files one .mif and other .mid .

Important Remarks :

1. The extreme of the arcs will be considered as nodes so if you want that two arcs share the same node is important to draw it like that. That's why the Fusion mode is important, when the pointer is over an already defined node it will change into a cross.

2. The style of the polyline is used to differentiate one-way and two-ways links. The two-ways links can have any style but one-way links must have style 2. Each two-way link in MapInfo corresponds to two one-way links in SCILAB. This correspondence is found in the sparse matrix Bidir.

3. The links can be given different colors, these colors will be in the output as the vector color, it can be useful to differentiate subsets of links.

EXAMPLES :

```
// MapInfo translation from the files SGL_Versailles_v.mid and
// SGL_Versailles_v.mif

[nodes,links,color,Bidir]=...
        MI2Scilab(MCS_DIR+'/examples/SGL_Versailles_v');

// Show the translated arcs in Scilab

MShowLinks(nodes,links)
```

SEE ALSO : `Scilab2MI` 83

13.4 MLogitB _ MULTI-MODE LOGIT EQUILIBRIUM (BELL METHOD)

CALLING SEQUENCE :

```
[FL,F]=MLogitB(A,D,ModeList,theta)
```

PARAMETERS :

`A` : list of nxn nodesxnodes matrices of travel times of each mode
`D` : list of nxn nodesxnodes matrices of demand flows of each class
`ModeList` : row list of the modes of each class
`theta` : stochasticity parameter
`F` : nxn nodesxnodes assigned flow matrix

`FL` : list of nodesxnodes assigned flow matrices of each mode

DESCRIPTION :
Computes the logit traffic assignment by the Bell method where all the routes are considered (not only the efficient ones).

For each mode m, to each route r on the graph defined by the matrix A_m is associated a weight $\exp(-\theta w)$ where w denotes the total cost on the route r for the mode m and θ is the parameter `theta`. The demands of each class c are defined by the matrix D_c and are assigned to a route and a mode, proportionally to their weights.

The graph must be strongly connected. If it is not the case we can add artificial arcs with large travel time in such a way that the new graph becomes strongly connected.

The parameter `theta` must be large enough to insure that the weight of routes with an infinite number of links is finite. When `theta` is too large numerical problems might appear associated to the limited length of floating numbers.

EXAMPLES :

```
// Graph generation (the graph must be stongly connected),
// n is the number of nodes, m the number of arcs.
n=10; m=40;
c1=m/(n*n);
A1=sprand(n,n,c1)+diag(sparse(ones(n-1,1)),1);
A1(n,1)=1;A1=A1-diag(diag(A1));
A2=sprand(n,n,c1)+diag(sparse(ones(n-1,1)),1);
A2(n,1)=1;A2=A2-diag(diag(A2));
A=list(A1,A2);
// Demand generation, p is the number of demand.
p=30; c2=p/(n*n);D1=sprand(n,n,c2);D1=D1-diag(diag(D1));
D=list(D1)
//theta definition (almost determinsitic)
theta=10;
// Flow calculation
[FD,F]=MLogitB(A,D,list([1,2]),theta)
```

See Also : `MAON` 69, `MLogitN` 73, `MAssign` 70

13.5 MLogitN — Multi-mode Net Logit Assignment

CALLING SEQUENCE :

```
MLogitN()
```

DESCRIPTION :
Computes the logit traffic assignment in the multi-mode case using the MLogitB or MAON algorithm according to the value of `%NET.properties.algorithm`. the travel cost is taken in `%NET.modei.mlinks.cost` and the assigned flow is put in the field `%NET.modei.mlinks.flow` of the variable `%NET` which is the Network tlist data basis which is global variable.

EXAMPLES :

```
// Graph generation (the graph must be stongly connected)
%NET=MTrafficExample('regular',4,1,1);
//Initialize the field %NET.modei.mlinks.time
// and %NET.modei.clinks.cost
%NET.carP.mlinks.flow=100;
%NET.carR.mlinks.flow=100;
%NET.busP.mlinks.flow=0;
%NET.busR.mlinks.flow=0;
Mlpf2();
//theta definition (almost deterministic)
%NET.properties.theta=3;
// Flow calculation
MLogitN();
MShowNet();
```

See Also : `MLogitB` 72, `MAssign` 70, `Network` 76

13.6 MTrafficExample — Multi-mode traffic network example

CALLING SEQUENCE :

```
%NET=MTrafficExample('name',p1,p2,p3)
```

PARAMETERS :

`%NET` : NetList of the traffic network
`name` : string to choose among 'Versailles', 'small', 'regular1','regular2'
`p1, p2,p3` : reals needed for examples: 'regular1' and 'regular2'

DESCRIPTION :
Generates a Network choosing among the following examples:

- `'small'` : a small test example.
- `'Versailles'` : a network corresponding to realist small example with 4 modes and two classes.
- `'regular1'` : a Network with 4 modes (carP, busP, carR, busR), two classes (P=Poor, R=Rich) corresponding to two values of time for a town with a 2D regular shape having $p_1 \times p_1$ nodes. If $p_2 = 1$ and $p_3 = 1$ there is only one demand for each class, if p_2 or p_3 are equal to $\alpha < 1$ there are αp_1^2 demands generated randomly for the corresponding class. If p_2 or p_3 are equal to $n_d > 1$ then there are n_d demands generated at random for the corresponding class. In this case the arcs that the buses can used are in the periphery and in the diagonal.
- `'regular2'` : Is the same that the previous case with the difference that the buses can use a subnetwork made from the network.

EXAMPLES :

```
%NET=MTrafficExample('regular2',4,1,1);
MShowNet()
```

SEE ALSO : `Network` 76, `MAssign` 70

13.7 Mlpf2 — MULTI-MODE LINK PERFORMANCE FUNCTION

NAME: :
Mlpf2 - Multi Mode Travel time functions

CALLING SEQUENCE: :

```
[]=Mlpf2()
```

DESCRIPTION: :
After updating the congestion field of `%Net` (`%NET.links.congestion`), it computes the travel time `%NET.modei.mlinks.time` and cost `%NET.modei.mlinks.cost` of each arc for each mode. The cost is defined precisely in the section of the Network tlist.

EXAMPLES: :

```
%NET=MTrafficExample('regular1',4,4,1);
%NET.carP.mlinks.flow=ones(1,length(%NET.carP.mlinks.t0))
%NET.links.congestion
%NET.carP.mlinks.time
```

```
%NET.carP.mlinks.cost
Mlpf2();
updateGP()
%NET.links.congestion
%NET.carP.mlinks.time
%NET.carP.mlinks.cost
```

SEE ALSO : `MTrafficExample` 74, `Network` 76

13.8 MShowNet — MULTI-MODE TRAFFIC NETWORK VISUALIZATION

NAME: :
MShowNet - Displays the current state of the network
CALLING SEQUENCE: :

```
[]=MShowNet()
```

DESCRIPTION: :
After updating the congestion field of `%Net` (`%NET.links.congestion`), it show in a *Scigraph* window the state of the network. A menu called **Class** allows to choose among the different classes or the global view. To continue in SCILAB the menu **Done** must be chosen.
EXAMPLES: :

```
%NET=MTrafficExample('regular1',4,4,1);
%NET.carP.mlinks.flow=ones(1,length(%NET.carP.mlinks.t0))
%NET.links.congestion
%NET.carP.mlinks.time
%NET.carP.mlinks.cost
MShowNet() // ''Choose Done when finished''
MAssign()
MShowNet() // ''Choose Done when finished''
```

SEE ALSO : `MTrafficExample` 74, `MAssign` 70

13.9 Network — MULTI MODE TRAFFIC ASSIGNMENT GEOGRAPHIC DATA BASIS

The database is stored in a SCILAB structure called tlist whose elements are again tlists. More precisely is

```
%NET=tlist(['Network','properties','nodes','links',
                  'ClassProperties','mode1','mode2',..],
                    prop,nodes,links,classprop,mode1,mode2,..)
```

where :

- `prop` : is a properties tlist,
- `nodes` : is a Nodes tlist,
- `links` : is the Links tlist,
- `classprop` : is a ClassProperties tlist,
- `mode1, mode2,...` : are TransportMode tlists,

It is built by the function Network

```
%NET=Network(nodes,links,classprop,mode1,mode2,..)
```

with the default properties.

PROPERTIES :

All the general properties concerning edition and computation are stored in the properties tlist defined by

```
tlist(['properties';'verbose';'algorithm';'JModel';...
       'NodeDiameter';'NodeBorder';'FontSize';...
       'ShowForbArcs';'tolerance',theta,Niter,N0,...
       ShowDemands,Show,bench],verbose,algorithm,...
       JModel,NodeDiameter,NodeBorder,FontSize,...
       ShowForbArcs,tolerance,theta,Niter,N0,...
       ShowDemands,Show,bench)
```

where : These tlists are built by the function.

```
Properties(verbose,algorithm,JModel,NodeDiameter,..
                   NodeBorder,FontSize,ShowForbArcs,tolerance)
```

The meaning of the fields and their defaults values are shown in the following table.

NODES :

```
tlist(['Nodes','nn','key','label','x','y'],nn,key,label,x,y)
```

Field name	Type	Default	Description
`verbose`	boolean	`%T`	Displays the intermediate results and performance of the algorithm
`algorithm`	function	`MLogitB`	The algorithm chosen for the assignment.
`JModel`	integer	`1`	The congestion function model.
`NodeDiameter`	integer	`30`	The node diameter used in the display.
`NodeBorder`	integer	`1`	The width of the node border used in the display.
`FontSize`	integer	`10n`	The font size used in the display.
`ShowForbArcs`	integer	`%F`	Whether to show or not the arcs with zero flow.
`tolerance`	real	`1.e-6`	The tolerance used to stop the algorithms.
`theta`	real	`1`	Stochasticity parameter for Logit assignment.
`Niter`	integer	`200`	Maximal number of iterations of the algorithm.
`N0`	integer	`0`	Starting n of the successive averages algorithm.
`ShowDemands`	boolean	`%T`	Whether to show or not the OD pairs.
`Show`	string	`'flow'`	Arc value (`'flow'`, `'cost'` or `'time'`) shown in the display.
`bench`	array	`0`	The intermediary results computed by the algorithm.

where :

- nn : is the number of nodes,
- key : is the integer-row of the nodes key (the default value is [1:nn]),
- label: string row of the nodes names,
- x : real row of the node x-coordinates,
- y : real row of the node y-coordinates,

A Nodes tlist is built by the function Nodes

```
nodes=Nodes(nn,key,label,x,y)
```

LINKS :
this tlist is defined by

```
tlist(['Links','key','tail','head','label','llength',...
       'capacity','congestion','glpp'],key,tail,head,...
       label,llength,capacity,glpp,occupancy,congestion)
```

where:

- key : is the integer-row vector of link-keys,
- tail : is the integer-row vector of tail node numbers of the links,
- head : is the integer-row vector of head node numbers of the links,
- label : is the string-row of link labels which can be empty,
- llength : is the real-row of link lengths,
- capacity : is the real-row of link capacities,
- glpp : is the real-2-row array of the parameters of the lpf function computing the links congestion contribution to the cost. The congestion function may have two forms according to the 'Jmodel' field `%NET.properties.JModel`, see the next subsection.

A Links tlist is built by the Links function

```
links=Links(key,tail,head,label,llength,capacity,congestion,glpp)
```

CLASSPROPERTIES :
This tlist is defined by :

```
tlist(['ClassProperties';'nc';'nm';'ModeList';'chi'],nc,...
        nm,ModeList,chi)
```

- nc : integer the number of classes, nm : the number of modes,
- ModeList : is a list of rows associated to the classes and containing the ranks in %NET of the modes used by these classes (for example list([1 2], 1) if there is two classes, the modes used by the first class are 'mode1' and 'mode2' and 'mode1' by the second class),
- chi : is a real nmxnm array, each row is associated to a mode and give the weights of the link performance individual mode functions used in the considered mode criterion.

It is built by :

```
ClassProperties(nc,nm,ModeList,chi)
```

TRANSPORTMODE :

The mode must be understood not only as standard transport mode (car, bus, train, etc...) but as a way to distinguish the uses of the links with different costs and/or travel times (two classes of passengers with different values of time using the bus or the car are considered here as four different transport-modes).

Each transport mode is defined by

```
tlist(['TransportMode','name','gcc','gci','vot','lpfmodel',...
     'mlinks','demands'],name,gcc,gci,vot,lpfmodel,mlinks,demands)
```

where

- name : is the name-string of the mode,
- gcc : is a real corresponding to the occupancy weight of the mode,
- gci : is a real weighting the link performance functions contribution with respect to the congestion and the travel time at empty congestion,
- vot : is the value of time for this mode,
- mlinks : is a ModeLinks tlist defining the link parameters depending of the mode,
- demands : is a Demand tlist associated to the demand-mode; in fact the demand depends only of the class and is defined as the sum of the demand-modes of the modes composing this class.

A TransportMode tlist is built by the TransportMode function

```
tm=TransportMode(name,gcc,vot,mlinks,demands)
```

MODELINKS :
To each mode is associated parameters and assignment results stored in ModeLinks tlist defined by

```
tlist(['ModeLinks','klinks','prix','t0','lpp','time','flow',...
       'cost','dflow',cc],klinks,prix,t0,lpp,time,flow,cost,...
       dflow,cc)
```

where :

- klinks : is the integer-row vector of the link-keys of the links used in this mode,

- t0 : is a real-row vector of the traveling-time on the mode when the network is empty,
- lpp : is an real-3-rows array defining the parameter of the link performance for the considered mode,
- time : is a real-row vector of the travel times associated to the computed traffic on the link,
- flow : is a real-row vector of the aggregated (on the commodities) arc flows of the mode,
- cost : is a real-row vector of the costs of using this mode, dflow : is a real-array of disaggregated (with the commodities) arc-flows of the mode, the rows are associated to the commodities,
- cc : congestion contribution

```
A ModeLinks tlist is built by the ClassLinks function
cl=ModeLinks(klinks,prix,t0,lpp,time,flow,cost,dflow,cc)
```

DEMANDS :
A demand is associated to each mode defined by

```
tlist(['Demands','key','o','d','v'],key,o,d,v)
```

where :

- key : is the integer-row vector of demand keys,
- o : is the integer-row vector of origin node numbers,
- d : is the integer-row vector of destination node numbers,
- v : is the real-row vector of demand flow values.

A Demands tlist is built by Demands function

```
demands=Demands(key,o,d,v)
```

EXTRACTING AND INSERTING DATA FROM A NETWORK :
As %NET is a typed list it is always possible to access to any field using the access functions, for example the x-coordinates of the nodes is given by %NET.nodes.x
COST FUNCTION :

The route costs perceived by the users are defined as the sums of the costs of the arcs. Given a class k and a mode m, they are calculated by

$$p_{am} + \nu_k \tau_{am} \tag{20}$$

where p_{am} is the price for an user of the class k (implicit in the mode) of using the arc a by the mode m; ν_k is the value of time for the users of the class k, and τ_{am} is the generalized time of an user of the class k using the mode m over the arc a. The value p_{am} is stored in `%NET.mode_m.clinks.price`, the value of time ν_k is stored in `%NET.mode_m.vot`, and the generalized time is the sum of three quantities: the free flow travel time (t_0), a linear combination of the travel time contribution (τ) of all the classes and modes and the congestion delay (J) :

$$t_0 + \sum_{j,\, n \in \mathcal{M}_j} \chi_{jn}^{km} \tau_a^{jn} \left(f_a^{jn} \right) + \rho_a^{km} J_a(\zeta) \tag{21}$$

The value of t_0 is stored in `%NET.mode.clinks.t0`. For the formula to compute τ there are 3 possibilities depending on the value of `%NET.mode_m.lpfmodel` : where c_a,m_a,b_a are

`lpfmodel`	Travel time function	Valid parameters
0	$m_a\, F/c_a + \max(0, m_a(F - c_a)^{b_a})$	$b_a \geq 1$, $c_a > 0$
4	$\log(c_a) - \log(c_a - F)$	$c_a > 0$
6	$c_a\, F + m_a\, F^{b_a}$	$b_a > 1$, $c_a \geq 0$

the row vectors of the matrix `%NET.mode_m.mlinks.lpp`.

The congestion delay J is an increasing function of the global occupancy ζ, a linear combination of all the flows over the given arc :

$$\zeta = \sum_{j,n \in \mathcal{M}_j} \sigma_{jn} f_a^{jn} \,. \tag{22}$$

The law of the function J depends on the value of `%NET.properties.JModel` and it can be

$$\left(\frac{\xi_a}{C_a} \right)^{\beta_a} \quad \text{or} \quad |\xi_a - C_a|^{-\beta_a}$$

where β_a is a positive real number and C_a is the practical or total capacity of the link a, these parameters are stored in `%NET.links.glpp`. The coefficients σ_{km} and ρ_a^{km} are stored in `%NET.mode_m.clinks.cc` and `%NET.mode_m.clinks.gcc` respectively.

COMPUTING THE ASSIGNMENT :

Assignments are done by means of the function `MAssign()`. The results for each mode `modei` are stored in the variable `%NET` in the mode fields `%NET.modei.mlinks.flow`, `%NET.modei.mlinks.time`, `%NET.modei.mlinks.cost`. And for the global congestion results `%NET.links.occupancy` and `%NET.links.congestion`.

SHOWING THE NET :

When a net is shown using `MShowNet()`, three types of arcs can be seen. The black ones, light-blue ones and cyan ones. The black and light-blue arcs correspond to the roads: - the light blue to the roads with zero flow, - the black to those with a non zero flow. The cyan arcs correspond to the OD-traffic demand, these arcs go from origin to destination.

See Also : `MAssign` 70, `MShowNet` 76

13.10 Scilab2MI — Scilab to MapInfo interface

CALLING SEQUENCE :

```
ok=Scilab2MI(net,name,nameout,disp_option,disp_option2)
```

PARAMETERS :

`net` Scilab variable containing the network
`name` the name of the MI source file without extension
`nameout` the name of the MapInfo table where the results will be saved
`disp_option1, disp_option2` displaying options.

DESCRIPTION :
This command is meant to display in MapInfo (MI) the results obtained for a network already recovered from MapInfo. It uses an existent MI file named `name` and creates two files for each mode/class (one .mif and one.mid) plus another two for the display of the global occupancy or congestion.

To see them in MI , they must be imported (table/import), then the values obtained in Scilab will be in the table associated with the file, and the width of the arcs will be proportional to the chosen magnitude (- flow, time or cost in the class/mode files , - occupancy or congestion in the global file).

The displaying options are used to specify the magnitudes shown in the arcs. The values for `disp_option1` are the strings `'flow'`, `'time'` and `'cost'`. They are used for the mode/classes values of the arcs. The values for `disp_option2` are the strings `'occupancy'`, and `'congestion'` and are used for the display of the global results.

```
      // Scilab to MapInfo translation
// 1) Choose the example Versailles where the scilab
// network is obtained from a MapInfo file
%NET=MTrafficExample('Versailles');
// 2) Make an assignment
MAssign()
// 3) Translate to mapinfo
ok=Scilab2MI(%NET,MCS_DIR+...
      '/examples/SGL_Versailles_v',...
```

```
      MCS_DIR+'/examples/man_example')
// 4) Look the content of MCS_DIR you need it in MapInfo
// 5) Open MapInfo and import the tables man_example*,
// look for them in MCS_DIR then in examples
```

SEE ALSO : MI2Scilab 71

References

[1] M.C. Bell "Alternatives to Dial's Logit Assignment Algorithm" Transp. Research, B, vol. 29B N. 4 pp. 287-295, 1995.

[2] M.G.H. Bell, Y. Iida "Transportation Network Analysis", J. Wiley & Sons, 1997.

[3] E. Cascetta "Transportation Systems Engineering : Theory and Methods", Kluwer Academic Press, 2001.

[4] CIUDADSIM: http://www-rocq.inria.fr/scilab/CiudadSim/.

[5] R.B. Dial: "A Probabilistic Multipath Traffic Assignment Model which Obviates Path Enumeration, transportation research, 5, 1971.

[6] S. Erlander, N.F. Stewart : "The Gravity Model in Transportation Analysis", VSP Utrecht 1990.

[7] C. Gomez (Editor) : "Engineering an Scientific Computing with Scilab", Birkhauser 1999 and (http://www-rocq.inria.fr/scilab/).

[8] T. Larsson, M. Patriksson :"Simplicial Decomposition with Disaggregated Representation for the Traffic Assignment Problem", Transportation Science, 26, 1992, 4-17.

[9] M. Patriksson : "The Traffic Assignment Problem, Models and Methods", VSP Utrecht 1994.

[10] MAXPLUS: ftp.inria.fr/INRIA/Projects/Meta2/Scilab/contrib/MAXPLUS/

[11] SCILAB: www-rocq.inria.fr/scilab/.

[12] SCIGRAPH: ftp.inria.fr/INRIA/Projects/Meta2/Scilab/contrib/SCIGRAPH/

[13] Y. Sheffi "Urban Transportation Networks" Prentice Hall, Englewood Cliff, NJ, 1985.

[14] P.A. Steenbrink "Optimization of Transport Networks", John Wiley & Sons, London, 1974.

[15] P. Toint, L. Wynter "Asymmetric Multiclass Traffic Assignment: A coherent formulation", *Proceedings of the 13th International Symposium on Transportation and Traffic Theory*, ed. J.-B. Lesort, Pergamon, Exeter, UK., 1996.

[16] L. Wynter "Contributions à la théorie et à l'application de l'affectation multiclasse du trafic", Thèse de doctorat de l'ENPC, novembre, 1995.

[17] L. Wynter "A convergent algorithm for the multimodal traffic equilibrium problem", INRIA Research Report, RR-4125, 2001.

Contents

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803